On Evaluating an Integrated Suite for ML-Agents Training, Visualization, and Debugging in Unity

Authors: David Grieco (dgri@itu.dk) & Toto Christensen (totc@itu.dk)

University: IT University of Copenhagen

STADS Number: KISPECI1SE

Instructor: Paolo Burelli

Date: 01/09/2023

Project Repository: https://github.com/Nenniana/MLA-Helper

Abstract

The ever-growing trend of including Machine Learning in the development and design process of video games has led to the birth of various libraries and tools over the past decade. These are often tied to a specific game engine and offer an easier way to access the potential of Artificial Intelligence. The common downside is a loss of freedom of customizability and a lack of transparency regarding the inner workings of their systems. One such system is ML-Agents, the most used toolkit for building ML-driven games in Unity. This project presents the testing and evaluation of an integrated suite for ML-Agents training, visualization, and debugging in Unity that enables users to better understand the inner structure of machine learning models. The project explores areas of Artificial Intelligence explainability, ease of use, debugging capabilities and usability.

Contents

Introduction

Machine Learning (ML) has gone through many eras, starting back to its allegedly first form, developed at the end of the fifties (Fradkov, 2020). In the early stages of Artificial Intelligence (AI), technology was limited, and the underlying theory was relatively undeveloped. Consequently, subsequent generations witnessed periodic surges in progress, often referred to as 'golden ages,' as well as intervals of reduced activity known as 'AI winters', as Fradkov explains. The fluctuating confidence in the evolving potential of early AI algorithms among scientists and businesses drove these fluctuations. Eventually, it all culminated in the current golden era, driven by computers' high computational capabilities and the ease of access to the latter. ML algorithms have become almost a staple in modern applications through integrations in the most commonly used software and engines (Kaynak, 2021).

Background Knowledge

At its core, ML is a subset of AI focused on analyzing vast data sets to make predictions and gain deeper insights into the systems producing this data. Among the foundational techniques in ML is Reinforcement Learning (RL), a family of algorithms intricately tied to the concept of rewards. In RL, agents initially explore their environments in a semi-random fashion. These interactions come with rewards, either positive or negative, meticulously assigned by developers to indicate the quality of outcomes. Over time and through iterative experimentation, positive rewards reinforce actions and behaviors aligned with optimal solutions, while negative rewards discourage erroneous decisions. This process, referred to as "training," ultimately yields models capable of making optimal decisions within specific input spaces. However, in highly unpredictable environments where the link between actions and progress remains elusive, even humans struggle to distinguish between good and bad actions. To address this challenge, the field introduced Deep Learning (DL), a technique harnessing neural networks with intricate architectural layers to train models using unstructured data autonomously. While adept at pattern recognition and issue identification in uncertain environments, DL cannot independently make

definitive decisions or progress toward optimal states. RL algorithms are seamlessly integrated with DL's neural networks to bridge this gap, giving rise to the concept of Deep Reinforcement Learning (DRL). In today's landscape, DRL is the predominant approach for training agents in dynamic, unpredictable environments, whether within video games or real-world scenarios. It remains the preferred choice for researchers and developers grappling with the complex challenges posed by unpredictable dynamic systems (Dargan et al., 2020).

Motivation

Digital environments with a high density of user interactions are a perfect fit for ML applications (Mousavi et al., 2018), which can enhance the experience, making it more realistic and unpredictable. Both Unity (*Unity Engine*, n.d.) and Unreal Engine (*Unreal Engine*, n.d.), arguably the most used engines in the field (Christopoulou & Xinogalos, 2017), have received support for ML in the form of packages, libraries, add-ons, and plug-ins; they enable the developer to set up and run sets of training on multiple ML-Agents without the need of in-depth knowledge of the underlying oft complex theory. The result is a decrease in development time and, therefore, an increase in overall productivity. However, one of the main issues with such tools lies in the options given to the user and the effort required to customize them to fit specific scenarios that venture away from intended use cases. Additionally, another downside is the lack of transparency regarding the inner workings of the underlying algorithms and data structures used by the tools. In cases where the user is not accustomed to ML theory, such a lack of transparency could be considered a positive feature, as the user can make good use of it without delving into the details. However, it can quickly shift into a negative factor when an expert user is trying to look into the current structure of the agent's brain to improve the training and, in turn, its results. At the heart of this project, guiding it from its inception to its final refinements, was the aim to alleviate some of the issues developers might encounter when working with Unity's ML-Agents package (*Unity ML-Agents Toolkit*, 2017/2023), a tool designed to facilitate the integration of ML algorithms with Unity environments. The

objective has been to create a suite of tools to facilitate the user's interaction with the package, from training a model to visualizing its structure and debugging its behavior. The presented suite has been named MLA-Helper to emphasize its intended purpose: providing aid when using the ML-Agents package. The MLA-Helper suite aims to shed the lack of transparency in the package to empower beginner and expert users alike to customize and optimize their model's performances based on a clear representation of its data and the suite’s debugging capabilities. The primary objective was not to create something groundbreaking or entirely novel but to provide a valuable specialized suite to assist individuals who may face challenges in comprehending and working with the intricate architecture of ML-Agent's artificial neural networks.

Problem Statement

As explained, AI has become extremely widespread, with ML algorithms and techniques gaining adoption in various domains. Unity's ML-Agents package has become a powerful platform for training ML models in virtual environments such as games, simulations, and beyond. However, adopting AI models in real-world scenarios demands high performance, transparency, usability, and effective debugging capabilities. The challenge lies in closing the gap between the complexity of AI algorithms and the need for comprehensibility, usability, and troubleshooting. The problem addressed in this thesis revolves around the lack of integrated tools and techniques in the Unity ML-Agents ecosystem that specifically target AI explainability, usability, and debugging. Furthermore, the current toolset for ML-Agents, while powerful, can be daunting for users with varying levels of expertise, hindering widespread adoption and collaboration.

Success and Evaluation

The success of this thesis will be determined based on the following integrated success criteria and evaluation methodologies:

- **Explainability Quantitative/Qualitative Evaluation:** Questionnaire-based evaluations will be employed to compare MLA-Helper's visualization against Zetane (*Zetane Viewer*, 2021/2023) and Netron (Roeder, 2010/2023) regarding AI explainability. Participants will provide feedback on the interpretability, clarity, intuitiveness, and ease of use of the visualizations provided. This evaluation will also gauge the comprehensibility and utility of the insights derived from the visualizations.
- **Performance Quantitative Evaluation:** The MLA-Helper suite will undergo quantitative assessments to measure its performance against state-of-the-art tools, such as Zetane and Netron. Metrics, including execution time, memory usage, and resource utilization during AI model visualization and configuration, will be compared. These evaluations will use diverse AI models to ensure a comprehensive assessment.
- **Workflow Quantitative/Qualitative Evaluation:** User experience questionnaires will assess the suite's impact on AI agent development using ML-Agents. Participants will be surveyed on their perceptions of debugging capabilities and usability, while task execution will be measured. This evaluation will provide insights into the suite's influence on the development workflow and the resulting agent behavior. Participants will be asked about their experiences with debugging and usability when using ML-Agents, both with and without the MLA-Helper suite.

Problems Clarification

The core issues addressed by this research include:

- **AI Explainability:** The challenge of developing visualizations and techniques that effectively communicate AI agent decision-making processes, behavior, and attention mechanisms, bridging the gap between complex algorithms and human comprehension.

- **Usability Enhancement:** The need to create user-friendly interfaces and workflows that simplify AI agent configuration, training, and deployment, catering to a wide range of user expertise to encourage broader adoption.
- **Debugging Capabilities:** The complexity of identifying and resolving issues within AI agents' training and behavior in dynamic environments, requiring innovative tools and strategies to enhance reliability and performance.

By tackling these challenges and meeting the success and evaluation criteria, this thesis seeks to aid in advancing AI explainability, usability, and debugging in Unity ML-Agents, promoting responsible and effective deployment of AI technologies.

Other Implementations and Games

Software applications have been developed with DRL transparency issues in mind, enabling users to observe different properties related to a model (Sadangi, 2022): Some applications can visualize the layers of their network with their input and output values such as Zetane as seen in Figure 1, some show the type of mathematical function that each layer applies (Roeder, 2010/2023) as seen for Netron in Figure 2, while some provide a clear view on the training process and offer graphs that track specific parameters related to the algorithms used (*TensorBoard*, 2017/2023). Although these tools may provide an excellent overview of what is happening in an agent's brain during and after training, and have been an ample starting point for researching AI visualization, they provide no aid in runtime analysis or environment-centric information.

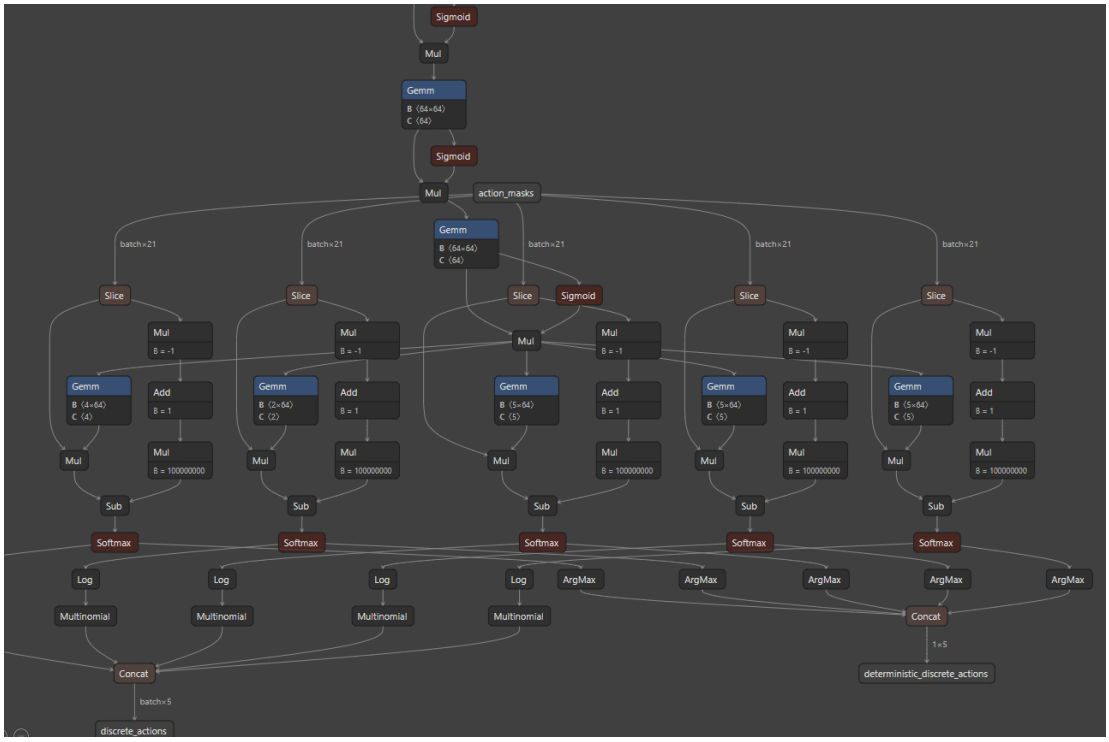
Figure 1

Zetane's visualization of the model's structure.

An interesting proof of concept was found in the work of Manjeet Singh about a visualizer tool for loading and running ONNX (*ONNX About*, n.d.) network models through Unity's built-in Barracuda (*Barracuda*, 2018) library (Singh, 2021/2023). Although it mainly focuses on visualizing convolutional neural networks, it still proved that ONNX models can be visualized in Unity during runtime as long as one has access to the underlying data they hold. It served as the first step into the initial phases of development.

Figure 2

Netron's visualization of the model's structure.



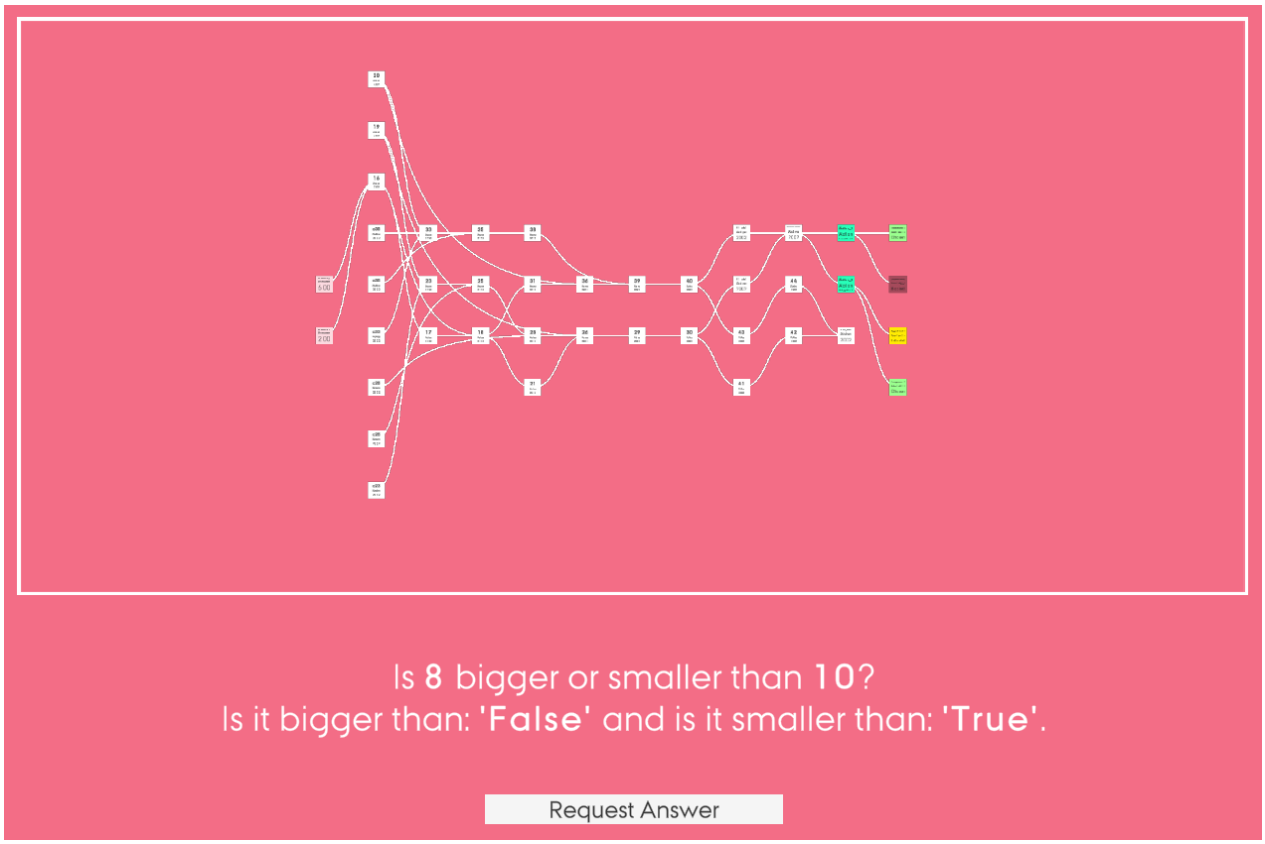
MLA-Helper

The MLA-Helper suite described in this paper has been ideated to fill in some of the gaps that other software, such as the ones described in the previous chapter, left unexplored. In order to achieve this goal in a reasonable amount of time, it was decided to focus on a subset of the features provided by previously mentioned applications. Specifically, the MLA-Helper suite enables the user to visualize the structure of the model, portraying its observation layer, hidden layers, and output layer. One of its diversifying aspects is the possibility of interacting with the visualization during the runtime within the environment the model is trained on, made possible by overlaying the model structure in the Unity scene view as seen in Figure 3. In the visualization, labels are given to each element to help the user understand their nature better. Action masks are visualized next to the actions in the output layer to display which actions are blocked based on the current observation space. An additional aspect found to

be missing in similar state-of-the-art tools is the inclusion of symbiotic elements that aid the user in an easier setup and training process while providing multiple means of debugging the resulting model. This finding led to the intention of giving MLA-Helper's workflow a usability and debugging purpose.

Figure 3

Visualization overlay in Unity.



Note. The white border seen around the visualization is the bounds of the overlay.

Background Theory

The following chapters will serve as a general introduction to the theory concepts touched by the thesis. These notions will help support the reasoning behind some of the choices made throughout the ideation and development process of the presented suite.

Machine Learning

At its core, Machine Learning revolves around teaching and modifying new or existing behaviors to machines through the use of algorithms that take into account what actions should be considered positive, helping the machine to reach its goal or negative, as in shifting it further from the objective. Formally, as defined by Arthur Samuel in 1959, Machine Learning is a field of study that provides learning capability to computers without being explicitly programmed (1959, as cited in Alzubi, 2018) or, as recently defined by Tom Mitchell et al.: "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E" (Mitchell et al., 2018). During decades of innovations, ML underwent massive changes and many different approaches and algorithms. However, these can be mainly grouped into six different categories based on their features (Oladipupo, 2010):

A. Supervised learning algorithms generate a function that maps inputs to desired outputs with the help of labeled examples set up by the developer.
B. Unsupervised learning algorithms model a set of inputs without any labeled examples.
C. Semi-supervised learning algorithms combine both labeled and unlabeled examples to generate an appropriate function.
D. RL algorithms learn a policy of how to act given an observation of the world. Every action in the environment has a consequence that guides the learning algorithm.
E. Transduction algorithms try to predict new outputs based on training inputs, training outputs, and new inputs.
F. Learning-to-learn algorithms learn their own inductive bias based on previous experience.

As mentioned before, the suite this paper revolves around allows for visualization of models that follow the ONNX format, which can be used in more than one of the categories above. Nevertheless, this project was conceived to understand ML-Agents' brains at a deeper level and provide the package with

higher transparency. ML-Agents provides mainly two types of training, DRL and Imitation Learning (Unity Technologies, n.d.-a). The first is a member of the RL algorithms family, and it is also the tool's main focus, while the other is a member of the Supervised Learning algorithms family. Given this paper's use case and its connections to RL, the notions regarding the other types of ML will not be detailed. The background knowledge will instead focus on explaining the theory aspects of this specific type. The preference for RL in gaming, as opposed to other algorithms, can be attributed to the inherent characteristics of this medium. In games, there is always an environment where either players or Non-Player Characters (NPCs) engage in interactions. These interactions frequently trigger changes in the game world's state, affecting the player's progress towards their objectives. This concept resembles how a RL agent learns and optimizes its policy (as discussed in the upcoming section). Although plenty more different learning techniques are applied to games (Furnkranz, 2001), this will be the specific one explored in this paper.

Reinforcement Learning

To properly understand the logic behind RL, one must first understand the problem it is trying to solve. This premise will first be presented to dive deeper into what the algorithms are constructed around to provide a better overview and grasp of the theory at hand.

Problem setup

A RL agent lives and interacts inside a defined environment, with its inputs visible to the latter and outputs resulting from the latter's actions (Y. Li, 2018). Formally:

> At each time step t, the agent receives a state s_t in a state space S and selects an action a_t from an action space A, following a policy $\pi(a_t|s_t)$, which is the agent's behavior, i.e., a mapping from state s_t to actions a_t, receives a scalar reward r_t, and transitions to the next state s_{t+1}, according to the environment dynamics, or model, for reward function $R(s, a)$ and state transition probability

$P(s_{t+1}|s_t, a_t)$ respectively. In an episodic problem, this process continues until the agent reaches a terminal state and restarts. The return

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \quad (1)$$

is the discounted, accumulated reward with the discount factor $\gamma \in (0, 1]$. The agent aims to maximize the expectation of such long-term return from each state. The problem is set up in discrete state and action spaces. It is not hard to extend it to continuous spaces (see Value Function). (Y. Li, 2018)

The extension for continuous spaces will be explained later on in the Value Function chapter, after having established the basic terminology needed to understand the theory.

Reinforcement Learning Terminology

Before describing the meaning of a value function, it is important to examine the meaning of the frequently used terminology for RL algorithms. The following is a list of the most common terms in the field:

- agent, a digital entity able to perceive and interpret its environment, take actions, and learn through trial and error following a policy;
- features, an independent variable in a ML model;
- reward, a positive or negative value that the agent receives on taking an action in the environment;
- goal, the objective that the agent is trying to accomplish through learning the most optimal sequence of actions to be taken;
- step, a time metric that corresponds to the time it takes for an observation and, in turn, an action to be taken by the agent;

- episode, a sequence of interactions between an agent and its environment, starting from an initial state and ending in a terminal state;
- epoch, a complete pass over the dataset during training, consisting of a fixed number of episodes played using the current policy;
- batch, a fixed size sample from the dataset, or existing sampled data, representing part of a pass through the full dataset;
- observation space, the collection of information about the environment that the agent is allowed to capture during its learning process;
- action space, the collection of actions the agent is allowed to take to interact with the environment;
- hyperparameters, a collection of labeled values that are used to set up and customize the training to fit specific requirements.

Value Function

At the core of the RL algorithm lie the concepts of reward and that of the value that each action holds towards the completion of the agent's goal. In this subchapter, an introduction will be made to the notion of the value function and how it plays a fundamental role in the implementation of any RL algorithm. As mentioned in the problem setup:

> The agent's goal is to find a sequence of actions that will maximize the return; in other words, the sequence of actions that will increase the sum of rewards during an episode or the entire life of the agent, depending on the task. If the task is continuous, the sum of rewards will be infinite, but it is still a solvable problem through the use of a discount factor, which lies in the *[0,1]* range. (Torres, 2021)

This is shown in the following equation:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + \cdots = \sum_{j=0}^{T} \gamma^j r_{t+j+1} \quad (2)$$

Where G_t is the discounted return at time step *t*.

Although the sum is still infinite, if γ < 1, then G_t will have a finite value, If γ = 0, the agent is only interested in the immediate reward and discards the long-term return and, if γ = 1, the agent will consider all future rewards equal to the immediate reward. Torres goes on to say that this formula can also be written in the recursive form:

$$G_t = r_{t+1} + \gamma(G_{t+1}) \quad (3)$$

Given a return G_t, an agent's goal is to estimate how good it is to be in a certain state (or how good it is to be in said state and perform a certain action) in terms of the return. As mentioned before, the return could depend on actions that are yet to be taken by the agent. This is why the estimate is defined with respect to particular ways of acting, called policies, and usually denoted by π. The previously mentioned estimate of how good states or state-action pairs are, is called value function (or V-function) for state estimates and Q-function for state-action pair estimates. The agent's objective is to train in order to refine these estimates so that the best state or state-action pair is picked at each step *t*. A more precise definition of the two must first be presented to better understand the relationship between the V-function and the Q-function. Formally,

the V-function also referred to as the state-value function or value function, measures how good it is for the agent to be in a given state *s* according to the return *G* when following the policy π. The following equation describes the value of said function:

$$V_\pi(s) = E_\pi[G_t | s = s_t] = E_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1} | s = s_t] \quad (4)$$

Where π is the policy, *E* is the expectation (for transition functions that are stochastic), *G* is the total return at each time step *t* starting at the state *s* at a time *t* and then following a policy π.

The Q-function, also referred to as the action-value function, measures how good taking an action a is from a state *s* under a policy π. It defines the expected return *G* starting from *s*, taking action *a*, and after that following the policy π. In equation form:

$$Q_{\pi}(s,a) = E_{\pi}[G_t|S_t = s, A_t = a] = E_{\pi}[\sum_{j=0}^{T} \gamma^j r_{t+j+1}|S_t = s, A_t = a] \quad (5)$$

(Torres, 2021)

Relationship between V-function and Q-function. For both equations, an expectation is defined because of the policy's stochastic nature when choosing the next action, given the current state. This stochasticity is formalized by the notation $\pi(a|s)$, describing the probability that a policy π selects an action *a* given a current state *s*. Naturally, the sum of the probabilities of all the outgoing actions from a state *s* is equal to 1. This notion can be used to reformulate the value of a state *s* in the following way:

$$V_{\pi}(s) = \sum_{a} \pi(a|s) \cdot Q_{\pi}(s,a) \quad (6)$$

In other words, the value of being in a state *s* can be thought of as the sum of all the values of being in that state and taking an action *a* (among all the possible actions *a* that can be taken from that state) multiplied by that action's chance of being taken. The previously explained theory leads to one of the best known equations in RL literature, being the core part of the majority of algorithms in the field, the Bellman Equation (Bellman, 1957). This equation decomposes the value function into two parts: immediate reward and discounted future values (Torres, 2021). This separation is helpful because it makes it easier to compute the total reward value. Instead of calculating multiple sums together, the value can be recursively evaluated by finding the optimal solutions of subproblems of the original one. In the following two sections, the V-function and Q-function will be rearranged in order to include the Bellman equation.

Bellman Equation for the V-function. As Torres explains, the Bellman Equation for the V-function can be recursively defined as follows:

$$V_{\pi}(s) = \sum_{a} \pi(a|s) \cdot \sum_{s'} P^{a}_{ss'}(r(s,a) + \gamma V_{\pi}(s')) \quad (7)$$

The equation, as anticipated before, breaks down the problem of finding the value of being in a state *s* when following a policy π into an immediate reward *r(s, a)* to which the discounted value of the successor state is added. The equation also considers the stochasticity of the environment, hence the inclusion of the probability *P* of ending up in specific states after the action is picked. In a less formal way, the value of being in a state *s* when following a policy π, is equal to the sum, for every action *a* that can be taken from state *s*, of the probability of taking that action following said policy times the sum, for every state *s'* one can end up after taking said action, of the probability of ending up in *s'* after *a* is taken times the immediate reward of *s* plus the discounted value of being in the new state *s'*.

Bellman Equation for the Q-function. Torres goes on to show that the same recursive principle can also be applied to the Q-function's equation, resulting in the Bellman Equation for the Q-function:

$$Q_{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}(r(s,a) + \gamma \cdot \sum_{a'} \pi(a'|s') \cdot Q_{\pi}(s',a')) \quad (8)$$

The equation shows that the value of being in a state *s* and taking an action *a*, is equal to the sum, for every possible state *s'* one can end up after the action *a*, of the probability of ending up in a state *s'* when taking action *a* from state *s* times the immediate reward of ending up in that state from that action plus the discounted sum of all the Q-values of being in the new state and taking an action *a'* times their probability of being taken. Given the relationship between the V-function and Q-function, the equation can be rewritten in order to calculate the Q-value in terms of the state-value function:

$$Q_{\pi}(s,a) = \sum_{s'} P^{a}_{ss'}(r(s,a) + \gamma V_{\pi}(s')) \quad (9)$$

Optimal Policy. As anticipated before, the goal of the agent is to maximize the total cumulative reward in the long run. The policy which maximizes the total cumulative reward is called the *optimal policy* (Torres, 2021). A policy π' is defined to be better than or equal to a policy π if and only if $V_{\pi'}(s) \geq V_{\pi}(s)$ for all states *s*. An optimal policy π^* satisfies $\pi^* \geq \pi$ for all the policies π. An optimal policy is guaranteed to exist but may not be unique, which means that there could be multiple optimal policies that share the same optimal value. For this reason, the optimal value function is defined as the one that yields the maximum value among all other value functions that have different policies:

$$V_*(s) = max_{\pi} V_{\pi}(s) \tag{10}$$

Here V_* is the optimal value function for the state *s*. It is equal to the maximum value, for every possible policy π, of the value function for the state *s*. Similarly, the optimal state-action value function indicates the maximum reward the agent is going to get by taking action *a* from state *s*:

$$Q_*(s,a) = max_{\pi} Q_{\pi}(s,a) \tag{11}$$

During his studies, Bellman proved that the optimal state value function in a state *s* is equal to the action *a*, which yields the maximum possible expected immediate reward plus the discounted long-term reward for the next state *s'* (Torres, 2021):

$$V_*(s) = max_a \sum_{s'} P^a_{ss'}(r(s,a) + \gamma V_*(s')) \tag{12}$$

He was also able to prove the same notion transposed into Q-function terms:

$$Q_*(s,a) = \sum_{s'} P^a_{ss'}(r(s,a) + \gamma max_a Q_*(s',a') \tag{13}$$

Deep Reinforcement Learning

The previous chapter introduced the concept and background theory for RL algorithms in general, but most often when training agents in games a specific subtype of learning is applied: DRL. DRL uses the opposite intuition of shallow learning, where instead of only having two data processing layers,

multiple hidden layers are employed to provide more complex problem-solving capacities. At every layer except the first one, the input is computed at each unit as a weighted sum of units from the previous layer. Afterward, through the use of nonlinear transformation or activation functions, a new representation of the input is obtained and passed onward to the next layer. Weights and biases are also added on links between units from layer to layer and updated through backpropagation of gradients in order to match their optimal values which will lead the agent to solve the problem at hand properly (Y. Li, 2018). When talking about RL, a plethora of algorithms can be brought up that make use of the previously presented theory. However, mainly three categories are needed to map them uniquely: dynamic programming, Monte Carlo, and temporal-difference learning (TD). The first group of algorithms are well-developed from a mathematical perspective but require a comprehensive modeling of the environment, which is almost never available. Monte Carlo methods (Metropolis & Ulam, 1949), while not having the previous constraint, are not suited for incremental computation such as batch learning. The last group, being TD algorithms, require no model and are fully stepwise incremental and episodic as well, which brought them to be the main choice for RL in games (Chadi & Mousannif, n.d.)

Proximal Policy Optimization

There are multiple approaches when it comes to applying TD algorithms using Neural Networks (NN) for storing learning data. However, one of the most natural and direct ways of doing so is through the use of policy gradient algorithms (PG) (Chadi & Mousannif, n.d.). These methods directly map states to actions via a learnable differentiable function approximator such as NNs, and their use has become widespread due to its balance between performance and comprehension (Wouter van Heeswijk, 2023). A famous and widely known algorithm from the policy gradient algorithms family is Proximal Policy Optimization (PPO) (Schulman et al., 2017). The core of this algorithm lies in sampling trajectories (series of actions picked one after the other) based on the current policy and estimating the advantage they bring to the agent in terms of estimated reward. This advantage can either be positive if the reward is

higher than with older policies or negative it yields a lower reward. The advantages at each policy update are used to change the policy parameters (which in NNs are the node's weights) to move closer to optimality. Since big updates would provide an incentive to venture too far from the existing policy, a variant of the PPO algorithm called Proximal Policy Optimization CLIP was introduced that clips the ratio between the current policy and the old one, ensuring that the difference between the two can never be too big. The objective function of the clipped variant is the following:

$$L^{CLIP}_{\pi_{\emptyset}}(\pi_{\emptyset_k}) = E_{\tau\sim\pi_{\emptyset}}\left[\sum_{t=0}^{T}[min(\rho_t(\pi_{\emptyset}, \pi_{\emptyset_k})A_t^{\pi_{\emptyset_k}}, clip(\rho_t(\pi_{\emptyset}, \pi_{\emptyset_k}), 1-\epsilon, 1+\epsilon)A_t^{\pi_{\emptyset_k}})]\right] \quad (14)$$

In (14), ø is a set of customizable parameters that can be changed in order to adjust rewards and steer the training towards better results. $E_{\tau\sim\pi_{\emptyset}}$ is the probability distribution of trajectories (sequence of states and actions over the time horizon). $A_t^{\pi_{\emptyset_k}}$ is the advantage estimate of the current action, if positive it indicates that the picked action is better then other possible actions at this state. $\rho_t(\pi_{\emptyset}, \pi_{\emptyset_k})$ is the importance sampling ratio, in other words, the probability of taking action a_t at state s_t following the current policy $\pi_{\emptyset}$ divided by the same probability but following the old policy $\pi_{\emptyset_k}$. This means that if ρ_t is greater than 1 the action a_t at state s_t is more likely to be picked by current policy rather than with the old one and vice versa. If the ratio is between 0 and 1 the same action is less likely to be picked by the current policy compared to the old one. This ratio is used as a straightforward way to estimate the divergence between the old and the current policy when the policy has to be updated. The clip operator, as explained before, is needed to prevent updates that are greater in magnitude than a defined interval $[1-\epsilon, 1+\epsilon]$, in order to maintain stability throughout the training process. The min operator, on the other hand, is included so that the final objective becomes a lower bound of the unclipped objective, which translates to the clipped objective being picked when the advantage is positive and when the updated policy diverges too much from the old one (to ensure stability). Conversely, when the advantage

is negative and the divergence between policies is high, the unclipped objective is picked. Specifically, in cases where a large update increases the probability of taking worse actions, this solution allows the update itself to rectify the erroneous direction by lowering the overall objective's value.

Weights, Biases and Activation Functions

When explaining DRL a mention was made about weights and biases being added to each layer, as control values that would help in training's optimization. In NNs data can flow from the input layer to the output layer thanks to activation functions that apply differentiable, nonlinear operators to the layers (Nader & Azar, 2021). In particular, for each layer in the network, its previous layer's neurons' values are first multiplied by the weights on the links connecting them to the neurons in the current layer. Consecutively, the bias values (consisting in predefined scalars) are added to these products, to then pass the results as the inputs of the activation functions. If the outputs of the activation functions are higher than a specific threshold, which is dependent on the type of activation function, the current layer's neurons activate and data flow progresses forward. In the aforementioned computation weights are multiplied by the neurons' values while biases are added to the product; this is important because, from a mathematical analysis perspective, it means that biases are the only elements in the equation able to shift the activation function in order to better suit the training goals or to prevent unwanted results.

Unity and ML-Agents

Unity is a cross-platform game engine developed by Unity Technologies to make game development accessible and more straightforward. It provides the user with a collection of built-in tools that form the base on which games can be created. On top of the already packaged features, Unity enables its users to browse an online asset store where the community of both experts and semi-experts releases new packages and assets for others to download, with a high percentage of free content. These added features, among others, have quickly led Unity to become a popular choice for teams and

developers in search of an affordable solution, becoming one of the most picked engines by game developers (Unity Technologies, 2021). The assets in the store range from 3D models to aiding tools, to plug-ins that unlock new possibilities in games, and, among all, AI tools. The most famous AI asset in Unity is the ML-Agents package. This open-source project offers different gaming-oriented simulated environments and tools to train AI agents. The toolkit already includes the implementation of the leading state-of-the-art training algorithms for DRL, such as PPO. It utilizes a semi-interface-based approach to alleviate the user from tackling the heavy background theory on which said algorithms are based. Its structure is divided into four main elements: the learning environment, the communicator, the Python API, and the Python trainer. The environment is a high-level element that includes the Unity scene and all elements within; the communicator connects the learning environment with the low-level Python API, and the latter interfaces with the Python trainer implementation that holds and executes the training logic such as training algorithms, model building, and updating, etc. When setting up the environment for training, users must pass relevant observations to the agent and incorporate what subset of actions the agent is allowed to take by defining an action space. Users must also customize the reward signal for the agent by defining what events and actions in the game yield a positive or a negative reward, hence steering the agent's model toward optimal behavior. The resulting model from the training follows a defined structure format called ONNX to standardize itself and open up to external uses in other software. The user can customize a configuration YAML file (Ben-Kiki & Evans, n.d.) that will influence the training methods and features of the agent. One of the more important ones is the num_layers hyperparameter, which defines how many hidden layers the NNs are going to have (*ML-Agents Docs*, n.d.). This hyperparameter is usually useful to reach a final structure that matches the complexity of the problem at hand by increasing the number of hidden layers to reach an optimal result. Nevertheless, these hidden layers are not the only layers added to the model by ML-Agents. Many activation layers — mostly Swish activation layers (Ramachandran et al., 2017) — and logic layers are added as well.

Unfortunately, the additional layers ML-Agents introduces are not documented, and their goal is hard to grasp when looking into a newly trained brain.

ONNX Format

ONNX is an open format built to represent ML models. It defines a standard set of operators, which can be considered the building blocks of ML and DL models. It further provides a common file format to enable AI developers to use models with various frameworks, tools, runtimes, and compilers (*ONNX Home*, n.d.). As mentioned in the previous chapter, ONNX is the format used by ML-Agents to create and store models. In ONNX, layers hold the values of each neuron in data structures called "tensors." Tensors are a generalization of vectors and matrices; that is, while vectors have one dimension and matrices have two, tensors can have any number of dimensions, including zero (scalar value). Mathematically, a tensor can be defined as a pair of sequences *(V, S)* where *S* is the tensor's shape (a list of non-negative integers) and V is a list of values with a length equal to the product of the dimensions in *S*. The shape of a tensor indicates how it can be represented in space. In other words, the shape's length is the tensor's rank (number of axes), and the elements of the shape are the number of elements present in each of the tensor's axes (*ONNX Documentation*, n.d.). Through the analysis of tensors in the ONNX model, one can observe the values on each neuron and the weights on each link connecting two neurons.

Scriptable Objects

In the course of developing the suite at hand, a requirement for a data structure to manage essential model-related information arose. This led to the decision to incorporate Scriptable Objects (Unity Technologies, n.d.-b), one of the technology options provided by the Unity engine. This section will briefly introduce their concept to better understand their use in the following Implementation chapter. Scriptable Objects are employed to hold diverse types of data efficiently. Rather than attaching unchanging data directly to a GameObject, which would cause each GameObject instance to store the

same data in memory redundantly, Scriptable Objects are used to store this data and are then referenced by the GameObject instances. This approach optimizes memory utilization since the data is stored only once. Scriptable Objects are treated like assets and can be easily searched and modified in the project's asset database, making it a fitting tool for saving unchanging data and accessing its reference in the Inspector view of a GameObject in the Scene. The centralized nature of data stored in Scriptable Objects further prevents user errors where values composing the data might be defined differently than in other GameObject copies that use it. By storing the data in a single asset, the only way the definition of said data can change is by changing the definition of the Scriptable Object asset itself. Owing to these attributes, Scriptable Objects were incorporated into the suite's implementation to manage data pertaining to the agent's observation space and its action space, the latter explicitly using action masks. This choice allows for convenient data sharing and facilitates access from all components of the suite's structure.

Related Works

The importance of aiding machine learning researchers and developers in their interaction with training models and understanding their structure is present in an ever-growing set of fields (Chatzimparmpas et al., 2020). Chatzimparmpas et al. produced an evaluation of the primary surveys that were run on machine learning helper tools. Their article laid the foundations for assessing the relevant works that share similarities with the MLA-Helper suite. The majority of the works found by this study show a major trend in focusing on CNN visualization tools. This underlined the difficulty of the search for similar solutions to the present suite. Some similarities were still found in their conceptual approach to explainability, usability and debugging. The authors described the work of (Amershi et al., 2011), where the web application CueFlik (Fogarty et al., 2008) was used to interact with the model inputs dynamically, similarly to how MLA-Helper allows for observation customization. The main difference lies in the limitations of MLA-Helper's interaction with the model's observation space.

Specifically, the presented suite only provides for customization of the observations fed to the model after completing the training. CueFlik, on the other hand, was used to enable an interactive learning process, with users customizing the ranking of input features, thus changing the outcome of the training. A better acknowledgment of MLA-Helper's nature was gained following the survey of Explainable AI visualization tools for DRL models presented by Choo and Liu. Notably, the authors group AI Explainability tools into three distinct groups: understanding-focused, debugging-focused, and refinement-focused tools (Choo & Liu, 2018). The first group includes tools to explain the rationale behind the model's decisions through visualizations of their structures and internal information flow. The second group comprises tools that help machine learning developers detect issues within a DRL model. The final group is focused on optimizing results through an interactive user experience that translates into the active learning of the model. Given MLA-Helper's characteristics, its membership is split among the understanding and debugging-focused groups. Relevant tools included in the understanding-focused group are ConvNetJS's visualization modules (*ConvNetJS*, 2014) for visualizing data propagation through layers and DeepVis' toolbox (Yosinski, 2015/2023) for portraying the activation map of filters in CNN layers. Despite their close connection with MLA-Helper's objective of increasing the interpretability of DRL models, the described tools are designed explicitly for CNN models. Regarding the debugging focus, the presented suite does not share any similarity with the tools mentioned by the survey, as their means of aid is mainly that of training monitoring visualizations. The study by Liu et al., which classified machine learning tools in similar categories to the ones explained before, offers an overview of other similar solutions to the one presented in this paper (Liu et al., 2017). Namely, a topological node-based visualization for CNN models was described, following the same reasoning of better interpretability that led to the topological ordering of the layers in MLA-Helper (Harley, 2015). Furthermore, a study by Yu and Shi proposed a different classification of visualization tools for DRL (Yu & Shi, 2018). This classification divided tools based on the target users: beginners, practitioners, developers, and experts.

Their goals were also assessed based on the category they fit in. Specifically, tools for beginners focused on teaching deep learning concepts through visualizations. Tools for practitioners aimed at visualizing the network architecture. Tools for developers centered around debugging the training process and tuning parameters. Lastly, tools for experts focused on AI explanations for the model's decisions. The MLA-Helper suite has elements that fit all the categories above, particularly structure visualization (beginner/practitioner/expert-centric tools) observation and action mask editing, and visualization (debugging-centric tools). Moreover, the initially intended feature ranking tool reinforced MLA-Helper's membership in the expert-centric tools group. Some of the relevant works mentioned in the paper are Tensorflow Graph Visualizer (*Tensorflow*, 2021) and ActiVis (Kahng et al., 2017) for visualizing the model's structure, DeepEyes (Pezzotti, 2017) for spotting irrelevant elements to the training and general debugging, and the Class Activation Map visualizer by Zhou et al. for feature relevance in CNNs (B. Zhou et al., 2016). The paper by Aamir et al. presented the implementation of the Caffe2Unity library, used to visualize a CNN model's inner structure, explorable with virtual reality in the Unity engine (Aamir et al., 2022). The authors also used Shapley values to assess the importance of input features, similar to the initially planned method for implementing the missing feature ranker tool of the MLA-Helper suite. The tools described in these papers inspired the definition of the presented suite's structure. However, no information on other machine learning suites was found when considering their connection with the ML-Agents package.

Method

As explained before, the goal of this project was to create a suite aimed at filling some of the holes left by some of the most common visualization tools for artificial NNs and to empower the built-in features of ML-Agents. This chapter will describe the process that led to the final design, from finding state-of-the-art software to the actual implementation. The methodology will also touch upon what type of evaluation was planned for validating the main properties of the tool and its different features.

Choice of Research Type

Before starting the development process, it had to be decided what type of features the suite needed that were not present in other similar software and which could aid the use of ML-Agents to offer a more complete landscape to anyone working with the package. A screening was run to gain information on the current state-of-the-art visualization tools. The screening process was further based on quantitative data gathered from the analysis of each tool's characteristics. The results of this assessment led to the definition of the features that the presented suite had to include.

Initial Data Collection

The analysis was jointly run on Google and Google Scholar by searching for the keywords "machine learning," "deep learning," "reinforcement learning," "video games," "visualization," "tool," "neural network," and "graph." This initially brought an overflow of possible results that did not match the desired features. In a second analysis, it was decided to introduce some main screening criteria: the papers or websites found through the search had to contain at least one mention of a visualization tool for NNs *(1),* and the latter had to be software accessible on the internet and not a private tool for personal use or a proof of concept *(2)*. While the second search yielded improved results, compiling a list of the most frequently utilized visualization tools for Deep Neural Networks proved to be more complex due to the limited incorporation of such tools within the papers. The final approach was searching listings of the most up-voted software in this category on tech websites found via Google and then confirming their frequent use in academic papers through a Google Scholar search. The primary listings taken as a model of reference were from Neptune's tech blog (Sadangi, 2022), Ashish Patel's publication (Patel, 2019/2023), and Towards Data Science's publication (McCloskey, 2022). It was decided to exclude software that did not have any academic mentions, which led to a final list composed of the following software: TensorBoard, Zetane, Netron, Comet, MLflow, NN-SVG, and Visual Keras. NN-SVG was eliminated from the list since it solely provided a visual depiction of a NN that had to be manually

constructed within the software using parametric settings. It could not produce network structure from the model itself, which was the primary focus of the presented project.

Results

After finalizing the list, each entry was individually examined, and their features were analyzed in comparison to the specific use case. The subsequent list outlines the primary findings identified for each entry in the research results.

G. TensorBoard is a complete and optimized visualizer for tracking trends and training-related data of PyTorch models. Its relevance to the project is given by the fact that it can also be used on ONNX models thanks to Tensorboard's integration with ML-Agents. Unfortunately, even though its features are the most complete in the list, it does not provide the user with a visual representation of the artificial NN.

H. Visual Keras is an open-source Python library that enables the visualization of the architecture of Keras models, offering multiple views depending on the type of model at hand. The downside of this software lies in its model type limitations, as it can not run any models other than the ones created in Keras.

I. Zetane and Netron are similar tools that focus on visualizing the architecture of ONNX models in an interactive and data-centric way. The two tools offer the closest visualization to the one initially imagined for MLA-Helper.

J. Comet is a visualization tool that focuses exclusively on interpreting Convolutional Neural Networks. Due to this, the software is already far from the use case on which this paper focuses.

K. MLflow is a widely used tracking software for ML focusing primarily on ease of access. Its wide range of API enables users of any common programming language used for ML to include its practical tracking framework and optimize the training process. As mentioned previously with

TensorBoard, MLflow lacks a visual representation of the model's net, drifting away from the primary use case.

Highlighted Features

The data collected showed a relatively high number of widely used tools in the field of ML visualization—however, some critical features the presented use case needed could not be found in the list. The most immediate one was a lack of depth regarding the visualization of a model's network and its information flow. Another important note, given the ML-Agents use case, is that the model resulting from the package's training feature is not transparent in the way it is built, and, being in the ONNX format, only a few tools can shed light on its inner workings; these tools will still yield a decent understanding of how the model is structured in the form of activation functions and layers but will not grant an intuitive, in-depth view on how these layers are internally organized.

Project Development Plan

After accumulating sufficient data on state-of-the-art visualization tools that aligned with the objectives of the presented suite, a project development plan was formulated. This plan outlined the necessary steps to create a functional and valid suite to address the requirements specified in the problem statement. First and foremost, the general logic flow of the suite's user experience was defined: the presented suite should intuitively enable the user to select and load a previously trained model through Unity's Inspector interface. After the model is loaded, the user has to easily be able to visualize the inner structure of the model in the Unity scene and navigate it through mouse controls. Additionally, user interaction with the suite has to meet a satisfactory level of usability in where setting up, and understanding the process was optimized. Moreover, the user should be capable of quickly selecting a specific action or environment state and gather visual data on what input features were the most relevant ones towards said action or environment state. Once the main flow of the suite was defined, the next step consisted of developing the suite in Unity. The following chapter will revolve around the

implementation and the suite's structure; hence, elements hereto will be omitted in this chapter. Once the suite was complete and functional, an evaluation was planned and carried out to assess to what extent the goals set by the problem statement were met by the MLA-Helper suite. The evaluation had to not only take into account the user-experience related aspects of the suite but also consider its performance data when under different levels of stress. This led to a joint type of evaluation, featuring both qualitative and quantitative analyses regarding a variety of different metrics, such as explainability, clarity, intuitiveness, ease of use, usability and debugging capabilities. The evaluation also took into consideration other closely related tools from the list presented earlier, in order to compare their ability to fulfill similar tasks to the presented suite's one. After the end of the evaluation process, data was collected form the analyses and results were extrapolated and computed in order to present them in the ‘Evaluation Results’ chapter.

Implementation

This chapter delves into the practical implementation of the MLA-Helper suite within the Unity environment. It provides insights into the development environment, project structure, the core classes and functionalities that make up the MLA-Helper, and shortcomings.

Implementation Features

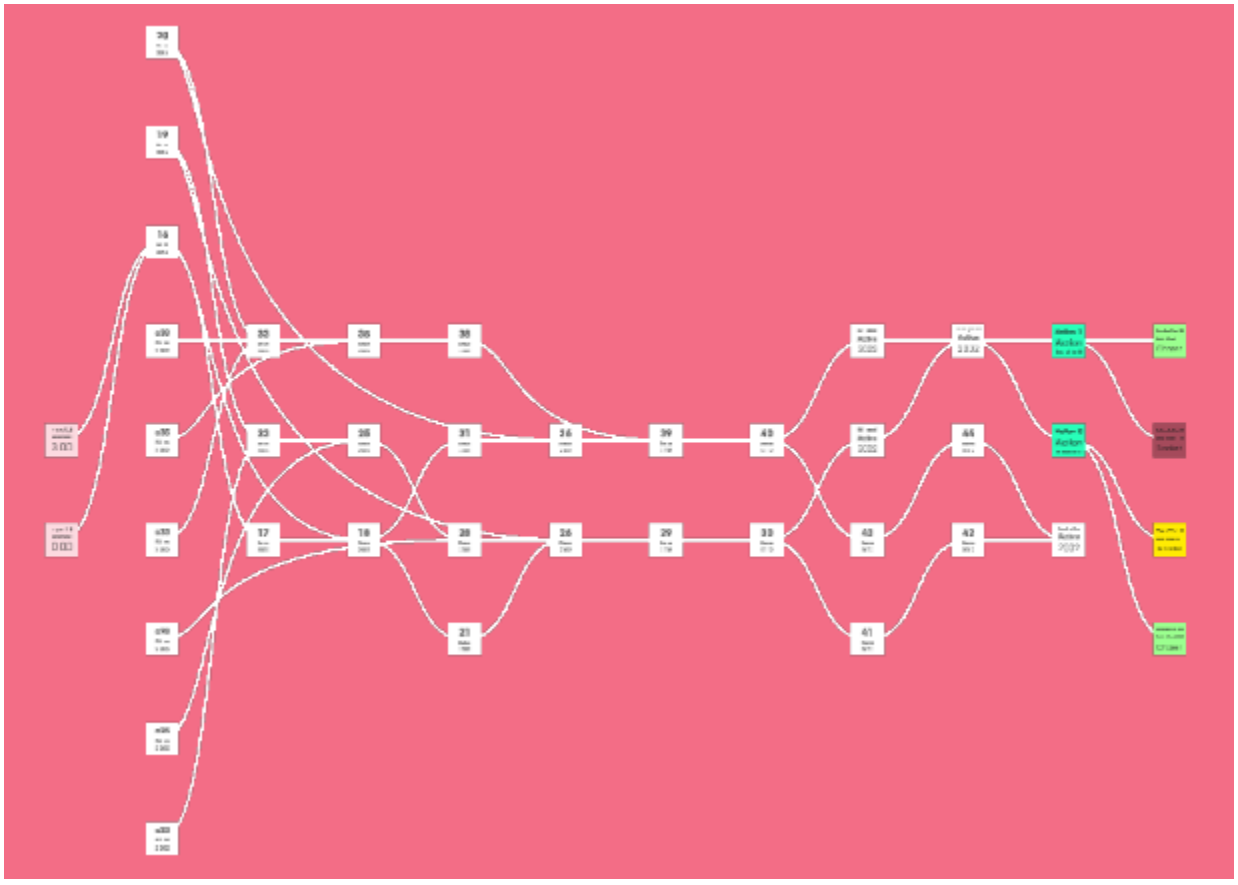
The MLA-Helper suite is focused on aiding the user in its interaction with the ML-Agents package. It achieves this through multiple interconnected elements of which a short description will be provided, before actual implementation details will be examined in the coming chapters:

- Visualization and Overlay: The visualization seen in Figure 4, offered by the suite portrays the model’s inner structure in a high-level manner during runtime. Firstly, the observation layer is detailed by visualizing its different labeled observations. This is meant to help the user visualize which element of the environment to which each refers. The following elements shown in the visualization are the hidden layers of the model, each topologically sorted. Finally, the action

layer is portrayed, with its actions visually divided into their possible value, each with its action mask information. This in-depth action mask visualization lets the user directly grasp which value of an action was chosen by the agent and which values were blocked/unblocked. The visualization is provided via an overlay onto the play window of Unity. This overlay enables users to view the model visualization on top of their current project, and allows it to be moved around, re-sized, zoomed and dragged.

Figure 4

Visualization of a small model in MLA-Helper.



- Global information and decoupling: The suite allows direct construction of the observation space and action masks. In the editor for the observation space, the user can set whether data for observation data should be constant or dynamic (taken from the environment). Additionally,

when creating actions, the user can set up action masks corresponding to each action, thus customizing the training logic. This information and data regarding other suite features are saved globally through Scriptable Objects, decoupling it while enabling it to be used within ML-Agent, MLA-Helper, and any element within the current project.

- Color coding: Color coding was included in the suite's visual components better to communicate the meaning of the model's elements. Notably, this feature shows which actions' values are blocked by action masks in the model visualization and conveys what action mask is active in the inspector. The intention is to let users interact with the suite more intuitively by having a visual indicator of the state of action masks and their effect on the agent's action choice, as well as dynamic vs. constant data.
- Deployment: The suite can be deployed into a project with minimal knowledge required. After the user accesses the deployment feature through the suite's interface, every component needed for its correct operation will be automatically added to the project's hierarchy.
- Reusability: Given the suite's nature as a collection of helper components, its reusability is an inherent characteristic. The suite can be used in different projects without dependency on their specific implementations.

Development Environment Setup

This chapter will detail a short description of the development environment in which MLA-Helper was created.

Code Editor and Extensions

The primary code editing tool chosen to develop MLA-Helper was Visual Studio Code version 1.81.1(*Visual Studio Code*, 2015) . The Unity extension version 0.9.0 (*Unity for Visual Studio Code*, 2023) for Visual Studio Code was used as it allowed seamless integration to Unity project files.

Collaboration

The project was hosted on GitHub (*GitHub*, n.d.) to support collaborative efforts.

Modification of the .gitignore File for ML-Agents. A modified .gitignore file was employed. This file was tailored to accommodate the unique requirements of the ML-Agents package and Unity, preventing the upload of unnecessary files and data.

Project Structure

This subchapter will touch upon directory hierarchy, asset types, scene structure, prefabs and GameObjects, as well as configuration files within MLA-Helper.

Directory Hierarchy

Below is the directory hierarchy within the MLA-Helper parent folder helping to maintain clarity and structure within the project:

- Images: Image asset for changing cursor textures is stored here.
- Settings: Scriptable Objects for MLA-Helper settings are stored here.
- Prefabs: Pre-designed game objects are organized here.
- Render Textures: The render texture for the visualization camera are stored here.
- Plugins: External packages and dependencies are grouped under this folder.
    - Sirenix (Odin Inspector): Assets and scripts in the Odin Inspector package.
- Scriptable Objects: Scriptable objects, used to store user action masks, observations, and collections hereof, are organized within this folder.
    - Action Masks: Scriptable action masks.
    - Collections: Scriptable Collections of observations and action masks.
    - Float Lists: Scriptable objects containing lists of float values.
    - Floats: Individual float values as scriptable objects.
    - Ints: Scriptable objects representing integer values.

        - Quaternions: Scriptable objects for quaternion data.
        - Vector2s: Scriptable objects storing 2D vector data.
        - Vector3s: Scriptable objects for 3D vector data.
- Scripts: All code files and scripts are organized into subdirectories.
    - Agent: Modifications to ML-Agents Agent class are stored here.
    - Helpers: Utility scripts and helper classes are subdivided into two categories.
        - Settings: Helper scripts for settings and configurations.
        - Static Helpers: Static utility functions and classes.
    - Inspector Interface: Script for the varying interface elements of MLA-Helper.
    - Model: Components and scripts for build MLA-Helper visualization.
        - Builder: Scripts related to building the agent's structure.
        - Structure: Components representing the agent's structure.
    - Scriptable Reference System: Scripts and components for managing scriptable references and objects.
        - Collections: Management of collections of scriptable references.
        - References: Individual scriptable reference objects.
        - Scriptable Reference Objects: Individual scriptable referenced objects.
    - Visuals: Scripts for managing visuals and visual information.
        - Layer Visual Information: Information about visual layers.

Asset Types

In the MLA-Helper project, the following asset types are employed:

- Scripts: All scripts are saved as .cs (C#) files.
- Prefabs: Prefabricated game objects are saved as .prefab files.
- Images: Image assets used for textures and visual elements are saved as .png files.

- Scriptable Objects: Scriptable objects are saved as .asset files.
- Fonts: Fonts used for text rendering are saved as .asset files.
- Render Textures: Render textures and related assets are saved as .renderTexture files.
- Model Files: Model files, specifically those in the ONNX format, are saved as .onnx files.

Scene Structure

When MLA-Helper is deployed within a scene, its structure is as follows. The scene structure is as follows:

MLA-Helper (Parent Object): This parent object contains inspector fields, managed through ModelConstructor, which are used to instantiate information for MLA-Helper.

A. Camera Render Canvas (Unity Canvas Object): The Camera Render Canvas serves as the canvas upon which the visualization camera renders, allowing for an overlay of the model visualization feature.

    A.1. Model Visualization Parent (Unity GameObject): This object is set to stretch across the entire length and width of the Camera Render Canvas and houses the ZoomDragUIWindow component script.

        A.1.1. Model Visualization Overlay: Specifies the bounds of the actual overlay.

            A.1.1.1. Border: Provides a visual border and allows for gathering OnDrag information.

B. Model Canvas (Unity Canvas Object): The Model Canvas is the canvas upon which the model is visualized.

    B.1. Model Visualization: Set to stretch across the entire length and width of the Model Canvas.

        B.1.1. Element Parent: Serves as a container for all layers and layer connection elements.

B.1.1.1. Layer Elements: Contains all layer information for the current layer.

a. Name (Unity TextMesh Pro): Represents the layer name.

b. Type (Unity TextMesh Pro): Denotes the layer type.

c. Value (Unity TextMesh Pro): Stores the layer value.

B.1.1.2. Line Element (Unity LineRenderer Element): Used for drawing connection lines between layers.

C. Visualization Camera**:** Used to render the Model Visualization and display it through the Model Visualization Overlay element.

Prefabs and GameObjects

In the MLA-Helper project, key prefabs and their respective components are as follows:

- Layer_Prefab: Used to visualize layer information and includes TextMeshPro elements for displaying layer name, type, and value. This prefab houses a child-class script of the LayerVisualInformation specific to the type of visualized layer.
- LineRenderer_Prefab: Incorporates a Unity LineRenderer component used for displaying lines within Unity projects. This prefab utilizes a LineRendererSmoother class, which, given a Vector3[4], creates a bezier curve to enhance the appearance of lines rendered by the LineRenderer.
- MLA-Helper_Prefabs: This prefab repository houses all elements needed during the deployment of MLA-Helper. The structure of this prefab aligns with the scene structure described in the Scene Structure subchapter.

Configuration Files

MLA-Helper provides a set of configurable files, all implemented as Scriptable Objects and stored as **.**asset files. These files are organized into three distinct groups, each serving a specific purpose:

- Settings:

- MLAHelperSettings: MLAHelperSettings is a Scriptable Object used to store general settings that govern various aspects of the MLA-Helper project. These settings encompass asset paths, element colors, startup procedures, etc.
- AgentHelperWindow: AgentHelperWindow serves as an intermediate storage entity for ML-Agents-specific settings during the deployment of MLA-Helper to a given scene.

- Scriptable Referenced Objects: Within this category, there are two subcategories: Action Masks and Observations.
  - Action Masks: Action masks store data related to individual action mask inputs for an ONNX model.
  - Observations: Observations encompass various data types, including floats, arrays of floats, ints, quaternions, Vector2s, and Vector3s. Scriptable Referenced Objects within this category store data related to observation inputs for ONNX models.
- Collections: This group consists of two types of collections: Action Mask Collection and Observation Collection.
  - Action Mask Collection: Action Mask Collection stores an array of action masks. These collections are to apply action masks as inputs for model visualization and ML-Agents components.
  - Observation Collection: Observation Collection stores an array of observations. These collections are used to provide observation inputs for model visualization and ML-Agents components.

Code Implementation

The code implementation in ML-Helper consists of four fundamental areas:

1. Model Visualization: This component manages ONNX model execution, constructs layer information, and presents it systematically to users.

2. ML-Agents Agent Class Extension: It facilitates data exchange between MLA-Helper and ML-Agents by extending the agent class.
3. Scriptable Reference System: This system establishes information classes to efficiently deliver observations and action masks to the previously mentioned components.
4. Deployment, Settings, and Central Hub: These encompass elements defining MLA-Helper behavior, deploying MLA-Helper into a scene, and setup of the central hub.

This subchapter begins with an examination of the model visualization code implementation, followed by the agent class extension. These segments form the foundational elements underpinning the scriptable reference system, providing a logical structure for exploring ML-Helper's code implementation. Lastly, elements to configure and deploy MLA-Helper will be examined.

Model Visualization

This chapter will focus on detailing how model layers are constructed by examining the supplied ONNX model, and how they are supplied through the system, to provide visualization for the user.

Onnx to Barracuda Model. ML-Agents save trained AI models in the ONNX format containing all layers, layer information, and tensors. However, to utilize these values within Unity, they must be converted to the Barracuda's neural net model data structure.

Barracuda Model Format: Barracuda is a deep-learning inference library, and its model format is well-suited for real-time execution within Unity. The format is designed for efficient inference on GPUs and CPUs. It represents NN architectures, layer types, and tensor data, allowing integration with Unity projects. Converting ONNX models to Barracuda's model format enables ML-Agents models to be used within Unity's real-time environment.

Model Construction. The ModelConstructor class' primary task is facilitating the transition from an ONNX model, as saved by ML-Agents, into a format compatible with Unity through Barracuda. This

process involves preparing the model with input data, including observations and action masks. Below is an overview of its key functions closely resembling their execution order:

ExecuteNewInput. The ExecuteNewInput function serves as the trigger for model execution. It manages the execution process, cleans up existing tensors, and initializes the LayerConstructor and ModelVisualManager on its first execution, facilitating model visualization.

CreateRuntimeModel. This function initiates the process by loading and converting the ONNX model into a Barracuda format model. It establishes a runtime model, laying the groundwork for subsequent operations.

CreateWorker. Once the runtime model is in place, creating a GPU/CPU worker is next. By utilizing the GetLayerNames function, this worker gains access to all layer names, ensuring full output during model execution.

BuildInputDictionary. To prepare the model for execution, the BuildInputDictionary function constructs an input dictionary. This dictionary will supply input data, including observation and action mask data, to the model during execution.

GetObservationTensor and GetAgentFloatArray. The GetObservationTensor function generates tensors for observations and action masks, considering their shapes and types. The GetAgentFloatArray function assists by determining whether to retrieve observation or action mask data based on specific criteria.

GetAgentObservationArray and GetAgentActionmaskArray. These functions construct the float arrays for observations and action masks, ensuring proper data organization and alignment with the model's requirements.

Constructing Layers. The LayerConstructor class, working in tandem with the ModelConstructor class, is responsible for constructing and sorting all layers required for MLA-Helper. These layers include observations, active layers, actions, and action masks. This subchapter outlines the key functions of the

LayerConstructor class, illustrating the logical progression from initial layer creation to updating layers in concurrent executions.

Constructor and Destructor. The LayerConstructor class is initialized with a reference to the ModelConstructor class to establish communication between the two. A destructor ensures proper event unsubscription.

Initial Layer Construction (OnModelStructureFinished). When the model structure is finalized, the OnModelStructureFinished function is triggered. It checks whether IndexedLayers, a layer repository, exists. If not, it constructs observation, active, action, and action mask layers. Otherwise, it updates existing layers.

Construction of Action and Action Mask Layers (ConstructActionAndActionMaskLayers). This function creates action layers and their associated action masks. It iterates through action masks by branch, constructing action mask layers and associating them with their respective action layers.

Construction of Observation Layers (ConstructObservationLayers). Observation layers are constructed based on the input dictionary provided by the ModelConstructor. The function iterates through input data, creating observation layers as needed.

Construction of Active Layers (ConstructActiveLayers). Active layers are constructed using the ModelConstructor's information of model tensor data. The function iterates through tensors, creating active layers.

Updating Existing Layers (UpdateLayerInformation). In concurrent executions, the UpdateLayerInformation function refreshes layer visuals. It updates observation and action layers, ensuring visual consistency with updated data.

Topology Sort. To ensure a clear and meaningful presentation of model layers, the ModelTopologicalDistanceSort class comes into play. This static utility class takes an array of ModelLayerParent objects and sorts them based on layer dependencies.

Depth Calculation (CalculateDepth). Layers' depths are calculated to signify their reliance on other layers. Observations and layers with no dependencies are assigned a depth of 0, indicating their foundational role.

Column Assignment. Layers are grouped into columns according to their calculated depths, establishing a depth hierarchy.

GridPosition Assignment. Each column's layers are positioned in rows, creating a logical and visually coherent order.

Model Visualization. The ModelVisualManager translates sorted model layers into visual elements within the scene, representing the model's structure. It constructs layer visuals, establishes connections, and ensures an organized presentation of the model's topology.

Initialization (Constructor). The ModelVisualManager is instantiated with essential parameters, including the layer constructor, prefabs for layers and lines, and a parent transform for visualization. It subscribes to the IndexedLayersReady event.

Visual Elements Creation (SetUpLayerVisualInformations). For each model layer, visual information is set up. This involves positioning the layers and constructing the corresponding visual prefab.

Bezier Curve Generation (SetUpLines). Bezier curves are generated to connect layers that have dependencies visually. These curves depict data flow between layers and ensure overlapping connections are distinguishable.

Layer and Line Instantiation (ConstructLines). Layer and line prefabs are instantiated in the scene to represent the model's layers and connections.

Layer Type-Specific Handling (CreateVisualLayerInformation). Visual components are created for different layer types, including observations, action masks, and actions. These components are initialized with the respective model layer.

Positioning and Parenting (CreateVisualLayerInformation). The visual elements are positioned at designated locations and parented under the provided transform for visualization.

Agent

An abstract class, MLAHelperAgent has been constructed that helps bridge the gap between the ML-Agents Agent class, which is the main interactable in the ML-Agents system, and MLA-Helper. This simple class ensures that observations and action masks from the supplied collections are applied correctly to the ML-Agents internals. This is done via WriteDiscreteActionMask and CollectObservations, which are overridden to allow easy data passthrough. The latter houses the AddObservation methods, which lays a partial foundation of the Scriptable Reference system, as it accepts only float values at its bare bones - which is taken advantage of in that system.

Scriptable Reference System

The Scriptable Reference system is the information delivery system within MLA-Helper and is the heart of the package in many ways. The system is built around Scriptable Objects and focuses on ensuring easy setup and delivery of observations and action masks by the user to the underlying systems that need them. The concept behind this system was developed by Ryan Hipple for games development, and has been adopted and heavily-modified for MLA-Helper (Ryan Hipple, 2017). The system allows for constant values that the user can set directly, to dynamic values set by applied systems and easily switching between these from the inspector. This subchapter will shortly examine the scriptable referenced objects upon which the system relies, the observation and action mask entities that reference them, and the collections that house them - as well as a short look at Odin Inspector, which makes vital additions to the system.

Scriptable Referenced Objects. As previously mentioned, scriptable referenced objects are scriptable objects. These information entities store float, float list, int, Vector2, Vector3, and Quaternion values for observations and bool values for the latter and for action masks. These are saved as assets and

are serializable by Unity, enabling easy storage of information that can be referenced at any point, from anywhere. These entities carry three values: a public bool 'UseConstant' from their parent class and a public 'Value' and 'ConstantValue' of their respective type. ScriptableActionMasks are unique, as they contain the previously mentioned values and public int values for 'Branch,' 'Index,' 'ConstantBranch,' and 'ConstantIndex.'

References. References are divided into two categories, observation and action mask references, and work in tandem with their respective scriptable referenced objects. The two categories both reference their respective counterpart, allow for their creation and for the manipulation of their values within the inspector and through script. These have a few elements in common, some of which are inherited from the abstract ReferenceParent class.

ToggleUseConstant. Enables toggling of the 'UseConstant' bool found in the scriptable referenced object through an inspector button, allowing the user to switch between using a dynamically set value or input a constant value themselves.

GenerateScriptableObject. Allows for the creation of a scriptable referenced object if one has yet been created for the reference in question and will set it as its ‘Variable’, allowing the reference to display and reference its information.

Rename. A button allowing the user to change the name of the scriptable referenced object comes to show if the user changes the name of its reference. As scriptable objects are assets, this is done after the name is final to ensure unnecessary calls to AssetDatabase and inspector updates.

ConstantValue. 'constantValue' is a private field that allows for displaying the scriptable referenced object's constant value and setting it through the inspector by the user. The value is kept private to ensure it is only set through the inspector.

DynamicValue. 'dynamicValue' is, as 'constantValue,' a private field, but allows for displaying the scriptable referenced object's dynamic value and has no setter, ensuring that the value is never set through the inspector. This makes the value display as read-only.

Value. The 'Value' field includes a setter for the scriptable reference object's dynamic value (Value) and a getter for either the dynamic or the constant value, depending on user settings. This is intended as the primary way a user sets or gets information through script.

Observations. All observations inherit from the abstract class GOReferenceParent (GO for Global Observation), among which the 'GetValue' float array field is essential. This virtual field allows getting the currently chosen value (dynamic or constant) as a float array for use with the ML-Agents 'AddObservation' system. This field is overridden for each child, ensuring their specific value type is returned correctly. All observations also inherit a 'GetObservationSize' used by observation collections to ensure the collection size equals the currently loaded model's observation space. This functionality increases in complexity for the 'ListGoReference,' where a public delegate allows the collection to subscribe to size changes for the list.

Action masks. The 'GlobalActionMaskReference' class ensures that action mask information is correctly set and displayed; this includes information about the action mask branch and index, which ML-Agents use to ensure action masks are correctly applied to actions. The class, therefore, includes an internal getter/setter for both fields reflecting the information in the associated scriptable referenced object. The class also includes a private bool, 'internalValue,' used in the getter for ‘Value’ to retrieve the current value. It is also employed with the 'ToggleActionMask' function to let the user see the current action mask value and toggle it easily.

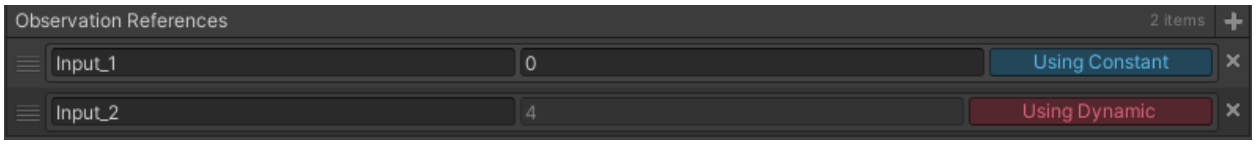
Collections. In order to enable users to create groups of observations and action masks and to allow easy input for both ML-Agents and the Barracuda model format, two collection types are created,

one for each distinct group. Specifically, the 'ObservationCollection' and 'ActionMaskCollection' classes include a public array of their respective types used in various systems as input.

ObservationCollection. The 'ObservationCollection' class's main functionality is to store all observation references used for a specific agent/model and to divide these into individual values for the 'LayerConstructor' class (Figure 5). This is done through the 'GetObservationPerIndex' function. The class further governs the functionality that examines the current observation size (all individual float values in the ObservationReference array) needed to ensure it equals the observation space of the model to which it will be applied. This functionality is primarily provided through the 'GetFloatArrayLength' function, but also using the 'OnCollectionChanged' attribute from Odin Inspector to catch 'ListGoReferences' added to the array and subscribe to their 'SizeChangedFromTo' delegate. The latter is significant as it allows updating the user, in the inspector, on the actual size of the collection and whether it equals that of the model.

Figure 5

An observation collection.



Note. This collection includes two float observations, the first of which is set to use a constant value.

ActionMaskCollection. The 'ActionMaskCollection' class stores all action mask references used for a specific agent/model and includes functionality to easily create a full array of references with created scriptable referenced objects based on user input, model name, and a path to save said assets (Figure 6). This functionality employs the static helper class 'AssetUtility' to ensure paths and folders are correct and unique.

Figure 6

An action mask collection.

Note. This collection includes four action masks, all using a constant value, where the third is blocked.

Odin Inspector Elements. Throughout the MLA-Helper project, Odin Inspector elements have been applied to ensure better visibility, less information clutter, and accessibility for users. This is especially true for the Scriptable Reference system, where Odin Inspector has been employed to hide or show labels and variable elements under specific conditions, ensure that collections and diverse elements are updated correctly, and visually communicate element significance in the inspector. This functionality could be attained with custom inspectors alone (upon which Odin Inspector functionality is built), but the package vastly improved this process.

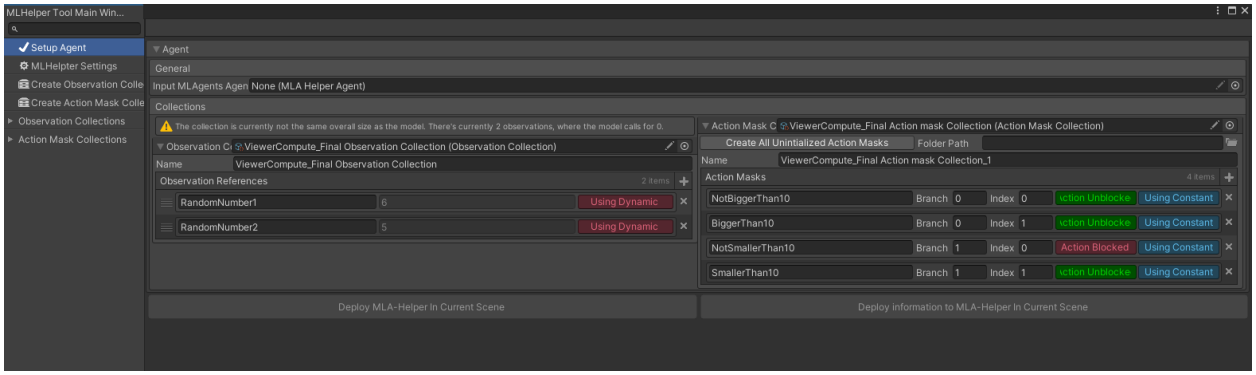
General

Hereby other important elements of a more generic nature within MLA-Helper, such as implementation of deployment of MLA-Helper within a new scene, information concerning its central hub, and its settings.

Deployment. The 'LoadAgentHelperWindow' class enables users to input an ‘MLAHelperAgent’ agent class entity, which is then used to load essential information and create an action mask collection of the appropriate size, as well as an observation collection (Figure 7). Once the information has been provided, users can then deploy MLA-Helper into the current scene or the information contained to an existing deployment. The class encompasses the following responsibilities and functionality:

Figure 7

Agent helper Page within MLA-Helper central hub.



Input and Parameters. Users provide input by specifying an 'MLAHelperAgent' agent component. The class retrieves vital information from the agent and its behavior parameters.

Action Mask and Observation Collection Creation. The class creates action masks and observation collections based on the agent's specifications. The action mask collection is sized according to the agent's action space, and the observation collection is initialized.

Validation and Warning. The class validates that collections match the model's specifications, issuing warnings if discrepancies are detected.

Asset Creation. Scriptable objects for action masks and observations are created and saved as assets in the project.

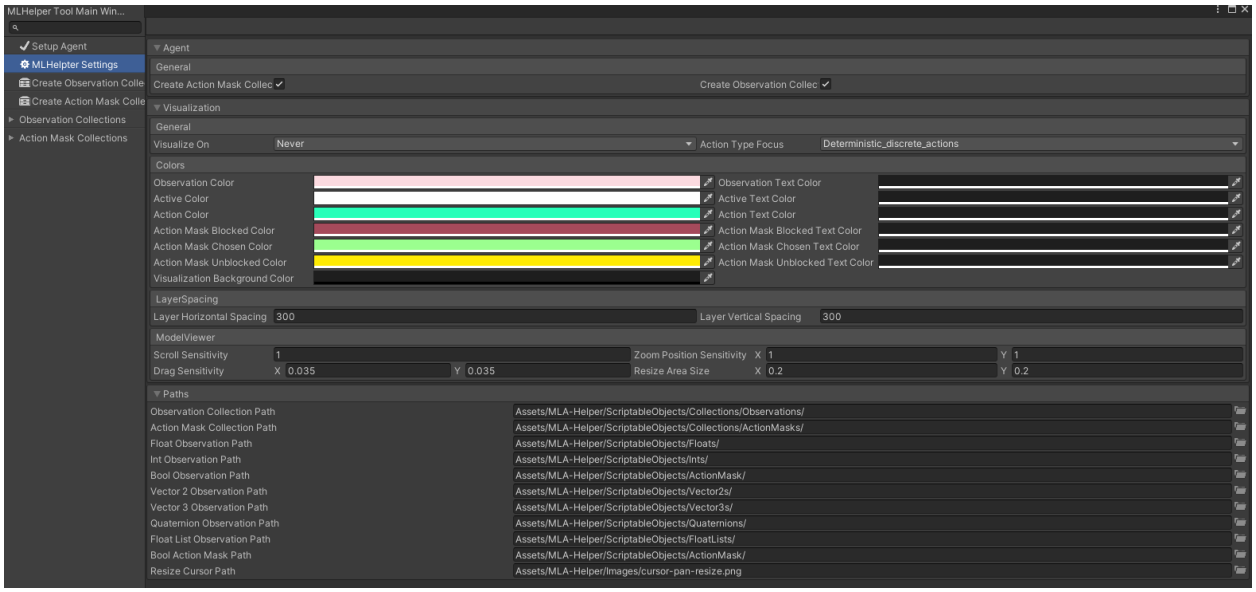
Deployment. Users can deploy MLA-Helper into the scene. If MLA-Helper is not present, it is created. If it already exists, the class deploys the updated collections and information.

Settings. MLA-Helper employs a scriptable object as a container for settings within the project. The 'MLAHelperSettings' include a private variable with a 'SerializedField' attribute for all applicable settings. Odin Inspector has further been employed to ensure the information appears organized and easily modifiable within the inspector (Figure 8). Each variable has a field with a private setter and a

public getter to allow other scripts to attain specific settings safely. A 'SettingsHelper' static class allows for easily attaining the settings object.

Figure 8

Settings page within the MLA-Helper central hub.



Central Hub. The 'MLHelperToolMainWindow' class sets up a menu item within Unity, providing a central hub for accessing various functionalities and settings related to MLA-Helper. This hub serves as the main entry point for settings, agent setup, and collections of observations and action masks for the user.

Menu Item Creation. A menu item labeled "MLA-Helper/Main" is created, which, when clicked, opens the MLA-Helper main window.

Menu Structure Building (BuildMenuTree). An OdinMenuTree is constructed, defining the menu structure and functionality. The menu includes sections for general settings, agent setup, creation of observation and action mask collections, and lists of all collections.

Collection Creation (CreateNew). A generic class, CreateNew<T>, facilitates users' rapid creation of new collections. Users can provide a name for the collection, and the Scriptable Object is created, saved as an asset, and assigned a name.

Collection Deletion (DeleteIfType). The class provides functionality to delete collections. When a collection item is selected, a deletion button becomes available. Users can delete observation and action mask collections directly from the menu.

Visual Elements and Functionality. The menu includes visual elements such as icons and buttons for user-friendly interaction. Search functionality assists users in quickly locating specific collections, observations, or action masks.

Cleanup (OnDestroy). Any created instances are destroyed upon closing the window to prevent memory leaks.

Performance Optimization

One vital focus point during the development process was to ensure a satisfactory level of performance during the interaction with the MLA-Helper suite. The following is a short rundown of implementation strategies, used technologies, and considerations hereto.

Optimizing Execution Timing

To enhance performance, MLA-Helper avoids utilizing Unity's Update and FixedUpdate methods. This approach ensures that code is executed precisely when required, reducing unnecessary computation and enhancing overall efficiency.

Utilizing the Burst Compiler for Expedited Model Setup

Incorporating the Burst Compiler was employed in mitigating particular bottlenecks during the instantiation of visual layer elements. It compiles C# code to highly efficient machine code, reducing execution time. This optimization significantly minimized processing delays, making the initial configuration of model layers more efficient and responsive.

Future Improvement: Leveraging Unity's Job System

Integrating Unity's Job System could further eliminate delay during initial instantiation and model setup. This system allows for efficient parallelization of tasks, potentially leading to substantial performance gains in instantiating visual layer elements. By leveraging the Job System, future iterations of MLA-Helper could significantly speed up the visual element creation process, optimizing the suite's responsiveness.

Shortcomings

Many decisions have needed to be made throughout the development of the MLA-Helper Suite, some based on tradeoffs, scope, and limitations, and some on uncertainty and lack of understanding. This has led to different shortcomings in MLA-Helper's implementation, which will be discussed in this chapter.

Purely Unity Based

Partly due to its nature, MLA-Helper brings shortcomings to the table. First and foremost, it is an editor-only Unity suite, meaning its use can hardly be extended to areas outside Unity development. The only way to use it in other environments is to convert a model to the ONNX format and create a new Unity project with the suite included where the brain can be visualized. The suite was mainly made editor-dependent to offer the ease of use of Unity's Inspector. A separate application could be developed using this suite's structure for inspiration. However, it would require better performance optimization and other utilities for visualization, such as shaders and low-level graphics programming, to surpass the current performances, which was deemed out of this project's scope.

No Training Statistics and Graphs

One of MLA-Helper's primary purposes is to visualize the inner structure of an ONNX model and give some insights into the agent's inner workings. However, it does not provide training statistics or graphs nor enable the interpretation of Convolutional Neural Networks.

Only Discrete Actions

The current implementation of MLA-Helper relies on an ONNX model, which regards Discrete Action spaces only. As such, MLA-Helper does not function in regards to a Continuous Action space, which ML-Agents supports. This limitation is a construct of how MLA-Helper employs action masks to label and identify action values within the model. A future update could alleviate this issue by re-organizing how action and action mask layers are implemented.

Missing Documentation

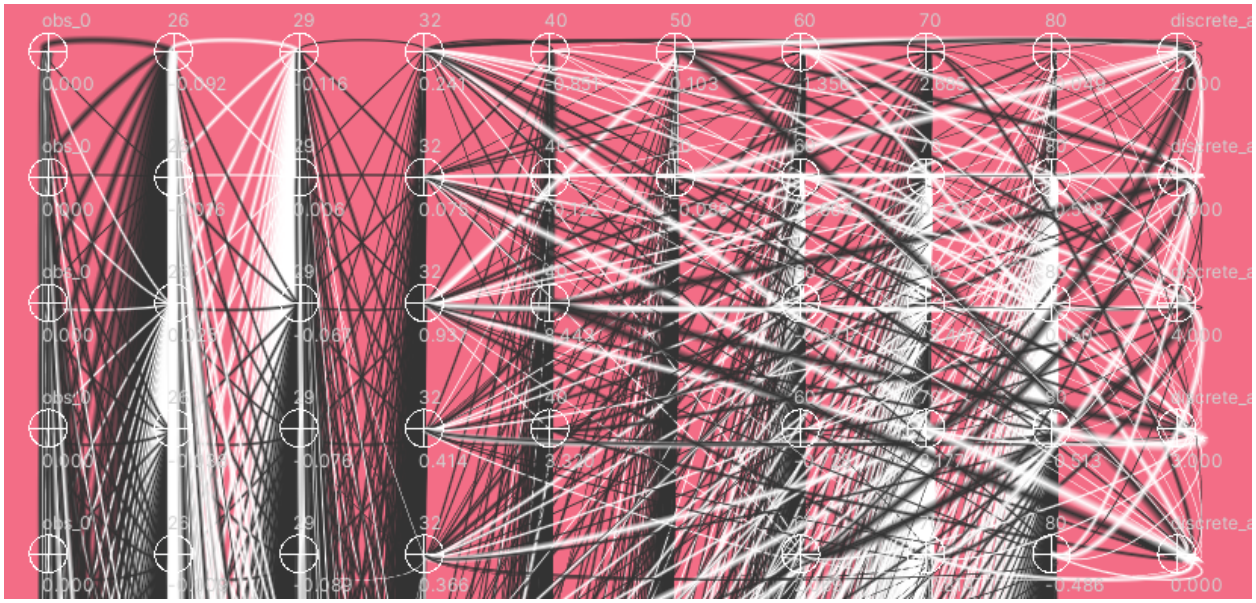
MLA-Helper provides little documentation on its setup procedure, use, or debugging steps besides its README.md on GitHub. This limited documentation can lead to erroneous use, cumbersome interactions with the suite, and limited adoption potential. This limitation was considered out of scope but would need remedying should the project be developed further.

No Display of Neurons and limited layer information

During the early development stages of MLA-Helper, all neurons within each layer and their links to other layers were visualized. As seen in Figure 9, this depiction cluttered the visualization while also resulting in performance issues. Based on this, the implementation approach was changed to focus entirely on the model's layers, disregarding all neuron information. While this change has allowed for a more organized model representation, incorporating neuron and layer information differently might have allowed users to attain other valuable information from the visualization.

Figure 9

Early version of model visualization



Note. This visualization depicts a relatively small model. The image is cut, showing only some neurons.

Information Clutter

Contrary to the previous shortcoming, one could argue that the current visualization implementation can also seem visually dense and that essential information can sometimes be difficult to discern. Enabling users to customize which layer elements are portrayed for a specific model might have alleviated this potential issue.

Proper Unity Package

While descriptively a Unity package, the MLA-Helper package is only an unregistered local package and does not take advantage of Unity's package system. This infers that dependencies, namely the TextMesh Pro, ML-Agents, and Mathematics packages, are not automatically added to the project when MLA-Helper is installed, among other disadvantages. This limitation stems from carelessness and a lack of knowledge of the package system, which rendered proper setup out of scope.

Unity Layer System

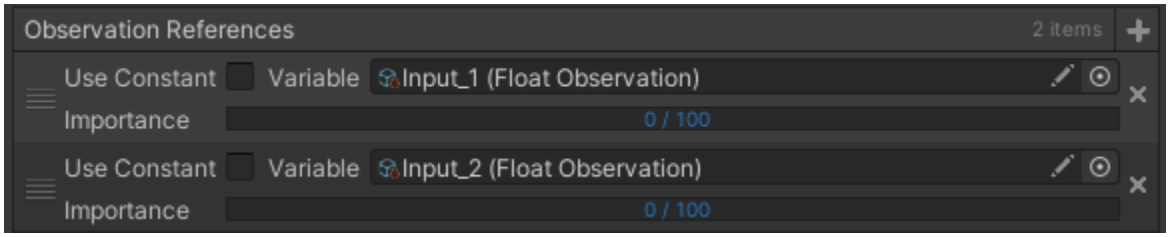
Unity's layer system has been employed to implement MLA-Helper visualization overlay within the Unity play window efficiently. However, as layers are project-based, and each project has a finite number of custom layers, this results in a setup process for MLA-Helper, which is more cluttered and error-prone than necessary. A solution to this difficulty could be taking advantage of Unity's Gizmo system and drawing the MLA-Helper visualization on top of the play window.

Missing Feature Ranking

The project's most significant shortcomings was the inability to complete the feature ranking tool, which led to its missing inclusion in the suite. This was due to an underestimation of resources needed for its completion and a lack of theory knowledge regarding class activation map algorithms that translated to the definition of its theoretical feasibility but not to its practical implementation. During the end of MLA-Helper's development, a possible methodology for its implementation was researched to ease its future inclusion in the suite. The feature ranking would measure the importance of a feature towards a specific action using Shapley values (Shapley, 1951). They measure the average marginal contribution of a feature across all possible coalitions of features, which could be used to rank the input features of the model by ordering them based on these values. In order to calculate their Shapley values, sampling could be employed to measure the feature's value at different times throughout the agent's interaction with the environment. The sampling would consider the specific action selected for generating the rankings of the features contributing to that action. Finally, once their values are recorded, the average marginal contribution of these features could be calculated through Shapley's formulas and displayed to the user, as seen in Figure 10.

Figure 10

Early version of the Scriptable Reference System



Note. Early versions of the Scriptable Reference system included a visualization of the importance attributed to the specific observation.

Evaluation

As previously stated, ML systems are becoming increasingly popular in all industries that can take advantage of them. Depending on the specific industry, though, different levels of responsibility are given to the machine, which could lead to dangerous scenarios or legal issues when the reasoning behind the outputs of a model or its inner workings are not feasibly explained. This topic is the core of a field called Explainable AI (Xu et al., 2019), consisting of methods for building tools that help explain the ML model's interactions with the environment. The field focuses on metrics for evaluating these tools to validate their usefulness. This chapter will touch upon metrics for validating ML explanation tools, since a complete overview of all the different types of explanation tools would be out of scope for this paper. Nonetheless, this chapter will showcase a taxonomy of the primary methods for explaining an AI model, pointing out to which category the presented suite belongs. Given the missing inclusion of the feature ranking tool due to its incomplete functionality, some of the categorizations will not be truthful to the current state of the suite. Nonetheless, their inclusion in this chapter will still prove useful for possible future releases of the MLA-Helper, complete with such a feature. On top of AI explainability, the evaluation will further assess the usability and debugging capabilities of the presented suite while also measuring the understandability, ease of use, and clarity of its visualization tool. In order to better

understand the metrics that will be analyzed, an essential mention of some of the key concepts of AI explainability will be made.

AI Explainability and Explanation Approaches

Explainability can be described as the extent to which a system can be explained to and understood by humans (J. Zhou et al., 2021). As Zhou et al. proceed to explain, the concept of AI explainability is made up of two main subcomponents:

- *Interpretability* describes the social interaction of *explainability*, meaning to what extent the explanation can be interpreted by humans. It can also have two additional properties: clarity (how unambiguous the explanation is) and *parsimony* (how simple and compact the explanation is). One more property regarding *interpretability* is *broadness*, measuring how generally applicable the explanation is.
- *fidelity (or faithfulness)* measures how well the explanation approximates the prediction of the black box model. It has two main properties, *completeness* (whether or not the explanation describes the entire dynamic of the model) and *soundness* (how correct and truthful the explanation is).

  (J. Zhou et al., 2021)

As noted at the start of this chapter, a relevant number of approaches in the field of Explainable AI can be categorized to better research or apply them thanks to some common features and characteristics. First and foremost, an AI explanation method can work at different points in the development and training timeline. In the first category, there are methods that are applied before the training, mainly focusing on the data that will be given as input to the training process. The second category refers to the methods applied during training which focus on making the model inherently explainable. The last group is made up of methods that are applied to trained models to grasp the meaning behind their choices better. These groups are respectively defined as *pre-model*, *in-model*, and *post-model* methods (J. Zhou

et al., 2021). Furthermore, methods can also focus on different types of explanations aimed at explaining every possible facet of the AI model's interaction with a task. Thanks to Arya et al. work on this categorization, the following is a solid list of categories in which AI explanation methods can be grouped based on their focus:

- Saliency methods: these methods highlight elements of the data to understand the model's task.
- NN visualization methods: their focus is to visualize the structure of a model's network, showing details about the data in it and helping the user understand its inner workings. Example visualization might be the layers of a NN, intel on their weights and biases, links between layers, etc.
- Feature relevance methods: these methods focus on showing the relevance between input features of a model and outcomes in the model's outputs. They can find relationships between specific features and outputs.
- Exemplar methods: methods that explain predictions of test instances using similar training instances.
- Knowledge distillation methods: these methods try to explain complex models through the use of other simpler models. In other words, another model is trained on the target model in order to simplify the explanation of its behavior.
- High-level feature learning methods: a family of methods that learn high levels of interpretable features using approaches such as Generative Adversarial Networks (GAN) for explanations.
- Methods that provide rationales: methods that generate explanations from input data but are local self-explanations. These methods might explain why specific inputs belong to specific classes, etc.

- Restricted NN architectures: methods that apply limitations and restrictions to complex models to make them easier to understand.

  (Arya et al., 2019)

A more general categorization of explanation methods is also given by Doshi-Velez and Kim, saying that these family of methods can be grouped into three main types:

- Model-based explanations refer to explanations that use a model to explain original task models. In this category, either the task model itself (e.g., decision tree) is used as an explanation, or more interpretable models are generated to explain the task model.
- Attribution-based explanations rank input features and use this ranking to explain the task model (feature importance or influence, etc.).
- Example-based explanations. This kind of method explains the task model by selecting instances from the training/testing dataset or creating new instances.

  (Doshi-Velez & Kim, 2017)

Taking these categorizations into account makes it easier to understand the nature of the presented suite. The current version of it, which does not include the feature ranking tool, has characteristics and features that place it within the NN visualization methods. However, if the feature ranking tool is eventually implemented in the MLA-Helper, it will also classify as being part of the feature relevance methods category. The connection with the NN visualization methods is due to the suite's ability to graphically portray the model's inner structure with its observation, output, and hidden layers. The characteristic that would put the latter in the feature relevance methods family, on the other hand, is the feature ranking tool's ability to highlight which feature from the input layer was the main contributor to the outcome action of the model's agent. Moreover, the suite can also be categorized through the taxonomy presented earlier from the works of Zhou et al., which places it among both the pre-model and post-model methods. This is because the suite facilitates the user in both setting up new agents for

training, along with their observation spaces and action masks, and gathering insights into the model once it is trained. Despite the usefulness of the Explainable AI methods, as Zhou et al. state:

> there are intrinsic issues in the previously listed methods, which can be summarized in the following categorization:
>
> - Statistical uncertainty and inferences: The previously explained methods provide benefits and explanations without quantifying the uncertainty of it, which could give a better understanding of its relevance to the user.
> - Causal explanation: ML models should reflect the true causal relations of their underlying phenomena in order to enable causal explanation; however, most statistical learning procedures reflect correlation structures between features instead.
> - Evaluation of explainability: There is no available way to quantify how interpretable a model is or how correct an explanation is.
> - Feature dependence: Extrapolation and correlated features can cause misleading explanations.
>
> (J. Zhou et al., 2021)

Given these issues, it becomes highly relevant that a framework for evaluating the validity of an AI explanation tool is defined to determine a tool's drawbacks and strengths in a transparent and exhaustive way. The following chapter will analyze some of the known approaches in the field of Explainable AI to serve as a starting point for the testing of this paper's specific use case.

AI Explanation Methods Evaluations

Explainability, as described beforehand, has at its core the concept of human understanding of systems. From this notion and the centric role that humans have in the interaction with AI explanation methods, a taxonomy was created by Doshi-Velez and Kim in order to better define the landscape for explanation approaches:

- Application-grounded evaluation: in this type of evaluation, the explanation method is tested in a real-world scenario with expert users; thus, performance with respect to that objective gives strong evidence of the success of explanations. An important baseline for this is how well explanations assist humans in trying to complete tasks, such as decision-making tasks.
- Human-grounded evaluation: in this simpler type of evaluation, the explanation method is tested by conducting simpler human–subject experiments that maintain the essence of the target application. The lack of need for expert users makes this the type of testing with the biggest potential participants pool, but the approach will depend solely on the quality of the explanation, regardless of the types of explanations and the accuracy of the associated prediction.
- Functionality-grounded evaluation: in this form of testing, no human participation is needed. It bases the testing metrics on some formal definition that is able to prove the explanation quality.

  (Doshi-Velez & Kim, 2017)

Another common, more general categorization with evaluation metrics consists of the binomial composed of *qualitative* and *quantitative* evaluations, both typically used to evaluate explanation methods (J. Zhou et al., 2021). Qualitative evaluation metrics are mostly related to questionnaires or interviews with open questions regarding usefulness, satisfaction, confidence, trust, etc. On the other hand, quantitative metrics include numerical data that can be retrieved from quantifying subjective opinions of testers. This can be done through well-defined scale measurements of human-machine task performance with readings taken from in-software scripts, or measurements externally applied during testing scenarios. These metrics might include the ones mentioned for qualitative evaluations but also more objective ones such as response time, accuracy, ability to detect errors, etc. Tied to the nature of

metrics, it is also important to consider their objectiveness, which can be divided into *subjective* and *objective metrics*. The first type revolves around the opinions of users on the ML explanation before, during, or after the task, and the second refers to objective information that can be analyzed and collected through tests that occur before, during, or after the task (Doshi-Velez & Kim, 2017). Both metrics are useful to validate an ML explanation, even though it may become harder to extrapolate a clear trend from subjective answers compared to objective measurements such as performance metrics, biological readings, or completion statistics. One more factor when preparing an evaluation of an ML explanation method is choosing the right type of task to validate the method at hand. In particular, two categories arise when categorizing tasks in ML explanation studies. The first, where studies evaluate the performance of humans employing ML to solve real-case tasks. The second, where proxy tasks are used instead to evaluate the humans' mental model of the ML system, with predictions and explanations, without necessarily evaluating how well the same users can perform said tasks (Doshi-Velez & Kim, 2017). Referring to the distinction presented in the previous section regarding model-based, attribution-based, and example-based explanations, a collection of objective metrics was defined by Zhou et al. to present different approaches in explanation methods evaluation. This collection analyzes potential values to measure, in order to assess the explainability properties of an AI explanation method. Due to this chapter focusing on the evaluation of the MLA-Helper and given its initially intended nature as a partial member of the attribution-based explanations family, only the details about the metrics regarding this specific type will be discussed. Before some of the metrics are analyzed, a better insight into the explainability aspect of the presented suite has to be given. As stated in the above paragraph, the suite's initially planned structure would have made it a partial member of the attribution-based explanations. This fact is due to its feature ranking aspect not entirely being part of the well-known family of methods (J. Li et al., 2018). In common feature importance methods, a ranking of the input features of an ML model is made based on their importance. The difference when comparing this

method with the feature ranking aspect of the presented suite is that the importance given to features, in the first case, is global and not dependent on a specific outcome. On the other hand, the latter would have elaborated a feature ranking for specific actions taken by the agent, highlighting which feature was the most important for a certain choice. Because of this difference, only a reduced amount of metrics from the collection could have been applied to the feature ranking tool, with some slight changes. A relevant one would have been *monotonicity*, defined by Nguyen and Martinez as the strength and direction of association between attributions (feature input rankings) and their expectations. This association is formally the Spearman's correlation between a feature's absolute performance measure of interest and its corresponding expectations (Nguyen & Martínez, 2020). The other metric that would have been central in validating the suite's ability to explain the importance of features that led to an agent's action is *non-sensitivity.* N*on-sensitivity is* used to ensure that zero-importance is only assigned to features on which the model is not functionally dependent. It is defined as the cardinality of the symmetric difference between features with assigned zero attribution and features on which the model is not functionally dependent (Nguyen & Martínez, 2020). These two metrics are indicative of how faithful a feature attribution explanation is (J. Zhou et al., 2021) and would have been the main quantitative metrics used to validate the MLA-Helper's ability to rank the importance of a model's features. Even though testing these metrics was not possible due to the feature ranking tool not being complete, the process behind their evaluation will still be detailed in the following chapter. Finally, the second member of the presented suite, the visualization tool, will be evaluated based on metrics that consider interpretability, focusing on general understandability, clarity, ease of use, and intuitiveness.

Procedure

This chapter will present a comprehensive evaluation of the paper's novel helper suite. The evaluation will be divided into three distinct parts: The first part will center around the assessment of the visualization aspect, divided into questionnaires and performance evaluations. Secondly, an overall

evaluation of the suite's workflow in contrast to ML-Agents independently has been conducted through a quantitative and qualitative questionnaire, as well as task execution measurements. Lastly, an assessment of the feature ranking element has been performed. Due to the feature ranking tool needing to be completed and functional, as explained in the 'Shortcomings' section of the 'Implementation chapter,' the quantitative and qualitative evaluations for testing its monotonicity and non-sensitivity metrics were unfortunately cancelled. The description of how they would have been organized if the functional requirements were met is still presented in this chapter to give an insight into possible future evaluations of the suite in case the feature ranking is finished.

Evaluation Method Motivation

The previous section touched upon the general aspects of the procedure that was carried out to evaluate the MLA-Helper suite. The primary approach used throughout the evaluation was using questionnaires with qualitative and quantitative questions. The questionnaires presented a higher amount of quantitative compared to qualitative questions. Qualitative questions allow participants to provide rich and detailed responses but lack consistency, which was needed to evaluate the metrics methodologically. The qualitative questions still provided insights into issues that the MLA-Helper suite might have and possible improvements that could benefit it. The method choice was guided by the questionnaires' efficiency in gathering user perceptions, interpretations, and opinions on a product (in this case, the MLA-Helper suite) in relation to their experience with it during previously run tests. The nature of asking both qualitative and quantitative questions provided answers that could measure the suite's impact regarding metrics via the Likert scale while also highlighting its strengths and weaknesses through open-questions (Rowley, n.d.). Another element that guided the choice towards employing questionnaires is their ability to gather data from different types of participants, expert and non-expert users. This ensured that a wide range of perspectives and experiences were captured. Questionnaires furthermore allowed for a structured data collection that provided every participant with the same list of

questions, facilitating consistency in the resulting comparative analysis and pattern recognition. Other methods considered during the evaluation's planning phase were observational studies, in-person interviews, the think-aloud protocol, and A/B testing. The motives for their exclusion vary and will be shortly detailed in the following list before moving on to the details of the evaluations:

- Observational studies. Observational studies can directly gather insights into a participant's use of the suite by observing their behavior and interactions. However, it presents multiple biases, such as information bias in misclassification of user behaviors due to faulty observations and confounder elements in user characteristics unrelated to their interaction with the tested suite (Jepsen et al., 2004). Questionnaires can overcome these issues by asking the participants to formulate opinions directly instead of relying on the testers' ability to observe the suite's impact on them.
- In-person interviews. In-person interviews were also considered to take advantage of the live testing but were later excluded from the evaluation to prevent inconsistent data from being gathered.
- Think-aloud protocol analyses. Think-aloud protocol analyses, consisting of gathering users' opinions while interacting with the suite, were excluded to prevent misinterpretations of the evaluated metrics. Although the protocol is valuable for gathering the immediate thought processes of participants, it would have been influenced by the participants' lack of familiarity with the suite's workflow. Another reason behind the exclusion of the think-aloud protocol was the lack of support from methodological research regarding its usefulness in testing usability (Van Den Haak et al., 2003).
- A/B testing. Finally, A/B testing was initially considered for testing the MLA-Helper suite with versions that presented differences, such as excluding color coding, labeling, and the action mask visualization. This type of evaluation would have provided more insights into the suite's

usability when lacking some of its features, validating their inclusion into the suite. The exclusion of the A/B testing was decided based on time constraints but could be applied in case the suite gets extended.

Due to these considerations, and that the difficulty curve of the suite is not an evaluated metric, questionnaires were instead employed to reflect participants' overall impressions better.

Evaluation of the Model Visualizer

As stated in the introductory chapter for the evaluation procedure, the assessment made on the visualization aspect of the suite included both quantitative and qualitative analyses. These analyses focused on validating the suite's ability to portray the inner structure of a ML model, its ease of use, its design, and its capability of yielding insights on possible patterns in the model's structure that may arise during training. A smaller portion of this evaluation made use of performance tracking. This performance tracking revolved around gathering data to assess the suite's overall performance. The following chapters will touch upon the main focal points of the validation and how the questionnaire testing group was sampled.

Evaluation Objective

The objective of the first evaluation was to assess whether the presented suite could provide an intuitive and comprehensible overview of the inner structure of the model. This assessment was further dissected into acquiring user statements on how well the suite's portrayed structure matched the user's mental image of it. Ease of use was also considered by questions regarding the user interface design constructed using Unity's Inspector view (Unity Technologies, n.d.-c). Given the split nature of the testing group, the questionnaire included questions aimed at providing insights into the helpfulness that the suite provided users (expert-related) and the potential didactic learnings (non-expert-related) that users

could grasp. The evaluation was run three times for each participant: in sequence, with either Zetane, Netron, or the presented suite.

Evaluation Participants

The first evaluation was run on a group of 21 participants, among which six were non-expert users, and the remaining 15 were expert users that had already used the ML-Agents Unity package in at least one past project. The sampling was carried out via the snowball method, consisting of relying on a population's subset of directly reachable participants and tasking them to reach other participants that meet the research criteria (Goodman, 1961). All of the participants were selected among the population of the IT University of Copenhagen where non-expert users had to meet the criteria of being IT students with a good knowledge of the Unity engine but no necessary knowledge of ML and ML-Agents. Snowball sampling was picked as the sampling method for reaching participants for the evaluation, given its cost-efficiency and cost-effectiveness, as the recruitment becomes recursive after the first group has been reached. Another characteristic that led to this choice is the speed at which snowball sampling can reach higher amounts of potential participants; given the limited amount of time available for the completion of this project, time efficiency had to be given the utmost priority. A notable advantage is also found in the trust established in the test group members reached by snowball sampling. As participants referred by other participants may be more likely to participate due to the trust established through their existing relationship. This can result in higher cooperation rates compared to recruiting individuals through more impersonal methods. Moreover, the ability to set well-defined requirements that describe the desired candidates, and are passed through to each participant, enables an effective search for specific sub-groups inside a broader set.

Purpose of evaluation and Anonymity. Participants were not informed of the direct purpose of the evaluation. This was done to minimize any potential confirmation bias that might result from participants' awareness of the evaluation's objectives. Anonymity was ensured throughout the

evaluation. Participants were assured that their feedback would remain confidential and that their identities would not be linked to their responses, promoting honest and unbiased feedback.

Questionnaire Structure and Evaluation Process

Once the required testing population was reached, the testing phase began. Before interacting with the tools, non-expert users were briefly introduced to some fundamental concepts of ML and artificial NNs, including the meaning of layers, neurons, weights, biases, and the intuition behind training a model with DRL. The non-expert users were also instructed on the ML-Agents package and its main features. After the introductory phase, the participants were presented with a sample model trained on a simple numerical game. The game consisted of two numbers displayed on a background. During gameplay, these two numbers were randomized within the range of 0 to 10. The objective of the game was to determine whether the sum of these numbers equaled, exceeded, or fell below 10. Once a choice was made, text appeared over the numbers, indicating whether the game had been won or lost. Participants were then tasked to open the model in Zetane, Netron, and the presented suite in their assigned sequence and observe the different visualizations for five minutes each. In order to reduce the bias that could have originated from the order in which the participants interacted with the tools, they were divided into three groups, each assigned with separate testing orders. The first group used Zetane first, then Netron, and then the presented suite; the second group used Zetane first, then the presented suite, and then Netron; the last group used the presented suite first, then Zetane and then Netron. After they interacted with the three tools, the questionnaire was given to each participant, and they were tasked to answer all the questions. In order to better contextualize the participants' answers, questions regarding their expertise were formulated; these questions included past usage of Unity, ML-Agents, and ML in general, years of experience with them, overall programming experience, and past usage of visualization tools such as Zetane and Netron. Another important note is the inclusion of a question regarding which tool sequence the participants experienced. By asking such a question to the

participants directly, the evaluation was able to prevent analyzing data rendered faulty by the participants not remembering to which tool they were referring. To understand the results of this evaluation phase, the questionnaire will be shortly detailed in the following section:

1. Kindly detail the order in which you interacted with the three visualization tools.
2. What aspects of MLA-Helper were particularly helpful in understanding the machine learning model's structure?
3. What were the differences (if any) in how you interpreted the information from the model using MLA-Helper compared to the other tools?
4. Were there any specific visualizations that you found particularly unclear or confusing?
5. What aspects of MLA-Helper's visualization do you believe could be improved for better clarity?
6. Are there any additional types of visualizations you would have liked to see?

On a scale of 1 to 5, where 1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, and 5 = Strongly Agree, rate your level of agreement to the following items:

1. The visualization I was presented with using MLA-Helper was more comprehensible than the visualization presented in Zetane.
2. The visualization I was presented with using MLA-Helper was more comprehensible than the visualization presented in Netron.
3. I found MLA-Helper's setup process more intuitive than Zetane's.
4. I found MLA-Helper's setup process more intuitive than Netron's.
5. I felt MLA-Helper's visualization of the model's structure matched the mental image I had in my mind more than when using Zetane.
6. I felt MLA-Helper's visualization of the model's structure matched the mental image I had in my mind more than when using Netron.

7. I felt MLA-Helper's visualization helped me understand more how information travels through the model compared to Zetane.
8. I felt MLA-Helper's visualization helped me understand more how information travels through the model compared to Netron.
9. Certain aspects of the visualization were unclear or confusing when using MLA-Helper.
10. Certain aspects of the visualization were unclear or confusing when using Zetane.
11. Certain aspects of the visualization were unclear or confusing when using Netron.
12. I felt the design of MLA-Helper's interface was easy to navigate.
13. I felt the design of Zetane's interface was easy to navigate.
14. I felt the design of Netron's interface was easy to navigate.
15. I was able to spot and understand the nature of logical patterns in the model's structure more easily when using MLA-Helper compared to Zetane.
16. I was able to spot and understand the nature of logical patterns in the model's structure more easily when using MLA-Helper compared to Netron.
17. I was able to translate the background theory I was given before the test into the visualization offered by MLA-Helper more easily than when using Zetane.
18. I was able to translate the background theory I was given before the test into the visualization offered by MLA-Helper more easily than when using Netron.
19. I was able to gain new valuable insights about the model's structure through the use of MLA-Helper compared to Zetane.
20. I was able to gain new valuable insights about the model's structure through the use of MLA-Helper compared to Netron.
21. I was able to gain new valuable insights regarding the model's decision-making process with MLA-Helper compared to Zetane.

22. I was able to gain new valuable insights regarding the model's decision-making process with MLA-Helper compared to Netron.
23. I believe MLA-Helper would benefit non-expert users interested in machine learning.
24. I believe MLA-Helper would benefit expert users interested in machine learning.
25. The insights from the visualization tool would increase my confidence in making informed adjustments to the agent's input features.

Quantitative Evaluation of Performances

Apart from user testing regarding the visualization tool, data was also collected with the goal of assessing its performance. The suite, Zetane and Netron were tested on three models of varying sizes: the first presented a total of 2 observations, 3 hidden layers with 16 neurons each, and 2 actions in the output layer; the second one presented 6 observations, 5 hidden layers with 64 neurons each, and 4 actions in the output layer; the last one, being the largest of the group, counted 12 observations, 10 hidden layers with 64 neurons each, and 8 actions in the output layer. The three tests were run on the same machine and the average measures relative to the evaluated performance metrics were gathered. The metrics used to validate the visualization tool’s performances against the other visualization tools were the following:

- Loading time. Measuring the time it takes for the tool to load and visualize the structure of the model starting from the moment the scene is started.
- Rendering performance. Tracking the frames per second (FPS), the engine stabilizes at when the visualization is running.
- Memory consumption. Monitoring the memory usage of the tool when visualizing the models’ structures.

Thanks to these metrics, the evaluation was able to give a broad overview of the performance of the suite under different levels of stress comparing its results with the ones obtained by running the tests on Zetane and Netron.

Evaluation of the Workflow

This chapter outlines the evaluation procedure designed to assess the impact and usability of the MLA-Helper package in the context of working with ML-Agents in Unity. The evaluation aims to measure its effectiveness in enhancing debugging capabilities and usability. Several key factors have been considered to ensure a rigorous evaluation, including using a control group, maintaining participant anonymity, concealing the study's direct purpose, and task execution measurement.

Evaluation Objective

The objective of the workflow evaluation can be split into the following assessments:

- Assess the Impact on Debugging Capabilities. The evaluation seeks to determine how MLA-Helper influences users' ability to identify and rectify issues within ML-Agents projects. It aims to quantify the degree to which MLA-Helper enhances software debugging capabilities compared to ML-Agents alone.
- Assess Usability. The evaluation will also employ the System Usability Scale (SUS)(Brooke, 1996) questionnaire to measure the ease with which users can work with ML-Agents through the MLA-Helper suite. This includes evaluating the intuitiveness of MLA-Helper's user interface and the simplicity of performing tasks with MLA-Helper compared to the experience without the suite. The SUS questionnaire will quantitatively capture participants' perceptions of usability, providing insights into the comparative ease of use between MLA-Helper and ML-Agents alone.

Overall, the evaluation seeks to provide a holistic understanding of how MLA-Helper influences users' experiences when working with ML-Agents regarding debugging and usability. By collecting quantitative

(Likert-scale and SUS) and qualitative data (open-ended questions), the procedure aims to view MLA-Helper's impact and gather valuable insights hereto comprehensively.

Evaluation Participants

For this evaluation, a participant selection process was conducted to ensure the evaluation was based on a well-defined and representative group. Participants were selected based on their prior experience with ML-Agents to establish a suitable baseline for the evaluation. A total of 15 participants were taken from the previous test group, specifically chosen to meet the criteria of prior ML-Agents experience (expert-users).

To minimize potential learning bias and ensure a robust comparison, participants were randomly assigned to two distinct groups:

- Group A (Experimental Group). In this group, participants received a standardized introductory course on MLA-Helper. This training aimed to familiarize them with the suite's features and functionalities.
- Group B (Control Group). Participants assigned to the control group received no additional training.

Evaluation Process

In this evaluation, participants were tasked with two primary objectives: configuring an agent from the ground up, capable of winning a game of rock-paper-scissors, and debugging an agent facing issues in a card-picking task. Each participant completed both tasks twice, but the sequence in which they employed ML-Agents alone and ML-Agents with MLA-Helper differed. This sequencing was randomly assigned to ensure unbiased results while efforts were made to guarantee an equal exploration of both sequences. Participants were given a Unity project that integrated both tasks.

Rock-paper-scissors Task. In the rock-paper-scissors task, participants worked towards setting up an agent capable of playing and consistently winning the game. Notably, participants were provided a

pre-trained model at the start of this task to expedite the process. The game's visual representation featured two hands displayed on a background. At the game's outset, the right-most hand visually selects one of three options: rock, paper, or scissors. Subsequently, the agent, represented by the leftmost hand, had to choose from the same set of options. After each selection, the game visually communicated whether the agent won, lost, or if the game ended in a draw.

Card-Picking Agent Debugging Task. Participants transitioned to the card-picking agent debugging task after completing the rock-paper-scissors task. In this scenario, participants were presented with an agent encountering issues related to incorrect input observations. The game involved selecting the pile with the highest sum from three piles of cards, each pile containing four cards. Card values were clearly displayed. Above the piles, the text explicitly stated the objective: "Choose the pile with the highest sum." Upon selecting a pile, the game provided visual feedback to indicate whether the agent's choice was correct or incorrect. The participants were tasked to debug the agent, fix the related issue, and ensure that the agent always picked the correct pile to win the game.

Upon completion of both tasks, participants were provided with questionnaires to gather their feedback on the debugging capabilities and usability of the suite as well as their overall experience.

Task Execution Measurement. To objectively assess the impact of MLA-Helper on task execution, the following metrics were recorded for each participant: time taken to complete agent setup of the rock-paper-scissors game and time taken debugging the agent in the card-picking game. This assessment is intended to evaluate MLA-Helper's usability and ability to assist in debugging through an objective measure.

System Usability Scale (SUS) Questionnaire. The System Usability Scale (SUS) questionnaire was included in the evaluation as a measure of overall usability. The SUS questionnaire consists of ten statements, each rated on a scale of 1 to 5. The inclusion of SUS allows for a standardized assessment of MLA-Helper's usability, providing valuable quantitative data.

Questionnaire Structure

Due to the utilization of the SUS questionnaire to evaluate MLA-Helper's usability, the explicit listing of questions presented to participants excludes all usability-related inquiries. The following are the questions given to the participants together with the SUS questionnaire, starting from the qualitative ones, followed by ones based on the Likert scale:

1. Have you noticed any specific debugging tools or features in MLA-Helper that you found particularly useful?
2. Did you encounter any issues debugging ML-Agents using MLA-Helper?

On a scale of 1 to 5, where 1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, and 5 = Strongly Agree, rate your level of agreement to the following items:

1. Using MLA-Helper significantly improved my ability to debug ML-Agents projects compared to using ML-Agents alone.
2. I feel more confident in my ability to debug ML-Agents projects with the assistance of MLA-Helper.
3. MLA-Helper's debugging tools and features are more effective than those available in ML-Agents.
4. Using MLA-Helper was not helpful when debugging ML-Agents projects.

Evaluation of the Feature Ranking

The third testing phase would have revolved around the suite's ranking of input features in relation to a particular action picked by the agent or a specific environment's state. Given this suite's technical and distinct nature, a quantitative evaluation focused on the monotonicity and non-sensitivity metrics was picked as the best candidate to validate its functionality and understandability. Moreover, smaller quantitative and qualitative analyses would have been possible to carry out through a questionnaire to assess the suite's utility. As noted at the start of the 'Evaluation' chapter, given the

missing feature ranking functionality of the suite, the evaluation will initially be presented to give an insight into how it could be carried out once the functionality would be implemented. However, no results regarding it will be discussed later on. A questionnaire was instead given to the previous testing group to assess whether this functionality would be valuable when debugging the decision process of an ML agent. The following sections will detail the objective of the initial evaluation for the feature ranking tool and how it was tackled once the tool's absence from the suite was confirmed.

Former Evaluation Objective

As mentioned previously, the focus of the third evaluation would have been to apply the AI explanation validation metrics. These metrics would have been examined to assess whether the feature ranking would have reflected a faithful representation of the actual importance of input features, given a specific action taken by the agent or an environment's state. As with the first evaluation, performance would have also been assessed in this third evaluation to prove how the feature ranking system performed in different scenarios. The evaluation would have also focused on the tool's ability to highlight crucial elements within the input domain for specific scenarios and the user satisfaction derived from this feature. The evaluation would have been carried out through a quantitative analysis regarding the faithfulness of the feature ranking and a questionnaire focusing on its usefulness and satisfaction.

Current Evaluation Objective

Since the feature ranking tool has yet to reach a functional state, the only assessment left consisted in analyzing the usefulness that such a tool could provide its users. The main focus of this evaluation was to determine whether a feature ranking tool would aid the users in debugging the decision-making process of their ML-Agents when interacting with environments. Another key point of the evaluation was assessing the confidence that such a tool would possibly bring users by confirming their knowledge of essential elements in the environments they created.

Evaluation Participants

The testing group for this third evaluation was formed by the same participants as the second evaluation, expert-users. This was decided due to the nature of the feature ranking tool, being much more particular and hard to grasp at first glance for non-experts. Participants took part in this evaluation after the second session to ensure they were well-acquainted with the tool before testing a specific feature.

Evaluation of Faithfulness

To determine if the tool accurately highlighted vital input features related to the agent's chosen action, concepts of monotonicity and non-sensitivity introduced earlier in the ‘Evaluation’ chapter would have been emphasized. To reiterate, monotonicity measures the strength of the association between two values, which in this case would be the input feature's actual importance and the expected importance defined based on the knowledge of the environment. On the other hand, non-sensitivity is used to validate faithfulness, measuring whether the zero-importance values are only assigned to non-important features for the model. By combining these two metrics, the tool's faithfulness validation was thoroughly planned. A non-zero monotonicity value derived from comparing the tool's rankings with constructed expectations would indicate a connection between the tool's output and the logical outcome based on environmental understanding, even if negative. Conversely, a zero monotonicity would suggest a lack of alignment between the tool's rankings and the importance attributed by experts to the same features, signaling a lack of faithfulness in the tool. The non-sensitivity metric, which measures the symmetric difference between features with zero attribution and those the model is not functionally dependent on, gauges how accurately the tool assigns zero-importance rankings to input features. An empty set resulting from this metric would imply that at least one feature rated as zero was, in fact, relevant for the action or environment state. This principle can also be applied with thresholds to identify when features near zero become relevant.

Cancelled Evaluation of Usefulness

An evaluation would have been carried out after the workflow evaluation to address a legitimate question regarding new tools or applications: even if a tool's validity has been proved already, is the tool solving a problem, or was the problem not there in the first place? To answer this question and assess the satisfaction gained from using the ranking tool, expert users of the previous evaluation would have been presented with another evaluation. In this task, they would have been introduced to the feature ranking tool and asked to try and formulate personal mental rankings of the input features based on their newly formed knowledge about the game. They would have then been asked to apply it to the same model used in their model visualization evaluation to view the feature in the process. After the experiment, all of the evaluation participants would have been asked to answer the questions of a final questionnaire that focused on this tool. The following are the questions that would have been present in the questionnaire mentioned above:

1. Did you encounter any difficulties or confusing elements while using the tool? If yes, please elaborate.
2. Can you describe any specific scenarios where the tool's insights could potentially lead to better decisions?

On a scale of 1 to 5, where 1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, and 5 = Strongly Agree, rate your level of agreement to the following items:

1. Based on my knowledge of the game, I found the tool's feature rankings to resemble the ones I formulated closely.
2. The tool would help me better understand how my models view their environments.
3. The information provided by the feature ranking tool about feature relevance is presented clearly and understandably.

4. The tool would be redundant because I already know the important elements in my games for achieving optimal results.
5. If I were to use machine learning, I would use this feature ranking tool to help me debug the behaviors of the agents.
6. I was satisfied with the tool's ease of use.
7. The insights from the feature ranking tool would increase my confidence in making informed adjustments to the agent's input features.
8. The feature ranking tool's insights enhanced my overall confidence in grasping the key factors influencing the agent's actions within its environment.

Current Evaluation of Usefulness

As mentioned, the previously described evaluation of usefulness was not carried out due to the missing feature ranking tool from the presented suite. Instead, a more subjective questionnaire was presented to the participants to attain knowledge on expert users' opinions regarding such a feature and its usefulness in debugging agents' decision-making processes and gaining confidence in environment knowledge. The participants were first introduced to the concept of feature ranking and how it would have been depicted by the tool. Subsequently, each expert-user participant was given a questionnaire. The following questions are the ones presented to the test group in this last evaluation:

1. Are there any aspects of the feature ranking tool's description you need clarification on? If so, please elaborate.
2. How could the feature ranking tool's insights potentially impact decision-making processes involving machine learning models?
3. Are there any potential limitations or challenges you foresee in the feature ranking tool's ability to highlight relevant input features accurately?

On a scale of 1 to 5, where 1 = Strongly Disagree, 2 = Disagree, 3 = Neutral, 4 = Agree, and 5 = Strongly Agree, rate your level of agreement to the following items:

1. I believe the feature ranking tool would help me better understand how my models view their environments.
2. I believe the feature ranking tool would be redundant because I already know the important elements in my games for achieving optimal results.
3. If I were to use machine learning, I would use this feature ranking tool to help me debug the behaviors of the agents.
4. The insights from the feature ranking tool increase would increase my confidence in making informed adjustments to the agent's input features.
5. The feature ranking tool's insights enhance my overall confidence in grasping the key factors influencing the agent's actions within its environment.

Potential Improvements and Considerations on the Evaluation

The evaluations described in the previous chapters tried to validate the suite's characteristics. Even though a mix of both objective and subjective metrics was employed, there is still ample space for improvement. The glaring issue was found in the limited number of participants able to fill out the questionnaires. Initially, a much larger test group was predicted. However, when the recruitment process started, it quickly became apparent how difficult it can be to find participants with a specific type of knowledge. A more extensive test group would have provided a more significant distribution of answers with smaller error margins and lower deviation standards. The exclusion of non-expert users from several evaluations was also an impactful decision. Although the evaluations where the exclusion was employed were centered around more technical aspects of the suite, the absence of data on how non-expert users would interact with these aspects partially invalidated the results. The evaluation of metrics such as usability and debugging capabilities could have benefitted from non-expert users'

opinions, increasing the overall completeness and generalizability of the process. Another lacking element was found in the scenarios the users were presented with: due to a lack of time, the environments the participants were introduced to was mechanically simple. This translated to the example environments not being adequate representatives of real scenarios to which the suite might be applied. A better approach would have been to also prepare more complex games, which would have helped examine the suite's features more deeply. Through these improved tests, the suite's capabilities would have been assessed in a more grounded way, which would also resemble its real-life applications better. The quantitative analysis could have also been improved upon by extending the data collection to more similar tools. Due to time constraints, this paper was only able to focus on the state-of-the-art tools that shared the most significant amount of similarities with the presented one. An objective evaluation of data from test runs on an extended list of visualization tools for ML would have put the presented suite's performances into a clearer perspective. Moreover, getting access to the source code or the low-level APIs of said tools to attach the performance tracking logic needed for the evaluation would have been much work, even with more considerable time and resources. Additionally, a sensitive aspect of the suite that was not evaluated due to the scope of this paper was its behavior when presented with ONNX models that were not trained through Unity's ML-Agents package. As stated throughout the paper, the presented suite is able to visualize any model in the ONNX format but was only tested with models that were trained using the previously mentioned package. A complete evaluation of said suite would need to include testing of major types of artificial NN structures to yield generalizable results. Finally, a self-explanatory improvement made clear throughout the evaluation description was the inability to run proper tests on the feature ranking tool due to its absence at the time of writing. If such a tool is completed in a second iteration of the suite, the detailed process for its evaluation will be employed, quantitatively measuring its faithfulness and usefulness.

Bias Considerations

In the preceding section, the discussion focuses on improvements within the evaluation process, highlighting the need to address specific biases present in the evaluation and questionnaire structure. Regarding the questionnaire, an observation was made concerning the potential influence of the MLA-Helper suite's prominence in comparisons. It was noted that the suite's mention as the initial element in these comparisons might predispose participants toward perceiving it as the preferred choice. Additionally, some questions were found to indirectly steer participants toward positive evaluations of the suite's features. In response to these observations, an effort was made to rectify this potential bias by introducing questions designed to elicit negative considerations during suite usage. This adjustment aimed to create a more balanced questionnaire that captures both positive and negative feedback from participants' interactions with the suite. Regarding the evaluation of the MLA-Helper's visualization tool, a deliberate variation was introduced in the testing order. The MLA-Helper's visualization tool was positioned as the first, second, or last tool in the testing sequence. This sequence manipulation was executed to assess metrics related to the MLA-Helper's visualization tool compared to other state-of-the-art tools. Notably, the primary focus was on evaluating the MLA-Helper's visualization tool against these other tools rather than comparing the latter tools themselves. Furthermore, a potential bias was acknowledged in the sampling method employed, namely snowball sampling. While this method effectively reaches more participants, its effectiveness depends on the clarity of the participant requirements conveyed to the initial batch. An initial selection bias is possible due to their close association with the test organizers. Additionally, this approach may inadvertently incorporate participants who only partially meet the requirements. Measures have been taken to mitigate participants' risk of aligning their responses with the developer's expectations (confirmation bias). Anonymity was assured in responses, and participants were explicitly informed that the provision of honest and impartial feedback was highly valued to account for this. Another concern relates to the

potential bias introduced by the introductory course in relation to the use of MLA-Helper, which might influence participants' perceptions of MLA-Helper (learning bias). To address this, a control group that does not receive the course has been included. By comparing the responses of this group with those of the trained group, an assessment can be made regarding the course's impact on participants' evaluations.

Evaluation Results

The previous chapter detailed the structures and methods used for the evaluations of MLA-Helper's characteristics, pointing out each evaluation objective and how the problem statement's metrics were assessed. This chapter is going to center around the resulting data that was gathered through both the qualitative and quantitative analyses. The chapter will be divided per evaluation in order to better focus the attention of the reader on a reduced amount of metrics at time.

Premise

Before proceeding to the analysis of the evaluation results, a premise on their validity is essential to their interpretation. As explained in the 'Evaluation' chapter, the quantitative analyses used Likert scale questions or adopted the SUS questionnaire (to assess usability). The SUS questionnaire has been validated through its extended use and solid results over the years (Brooke, 2013). On the other hand, researchers have discussed different characteristics of Likert scale questions. One central dividing conviction is whether or not Likert data should be treated as interval or ordinal. Some researchers argue that Likert scale data should be treated as both interval and ordinal based on the requirements of the study (Statistics Café, n.d.), while others argue that the nature of the data should be assessed based on the type of questions asked in the questionnaire (Boone Jr. & Boone, 2012). In particular, Boone Jr. & Boone present a methodology for grouping questions into two categories: Likert-type and Likert scale. The first category includes questions focused on different unrelated metrics, while the second group comprises questions that assess the same metric. Based on this difference, they treat the resulting data

with different statistical elements and methods for measuring central tendency, variability, associations, and statistical significance. Notably, Likert-type questions are analyzed through the median and frequencies of the results, while Likert scale-type questions are analyzed through the mean and standard deviation of the results.

Evaluation validity

In the case of the presented study, Likert-type and Likert scale-type questions were used. However, due to the structure of its questionnaires, only some of the metric assessments fulfilled the requirements for testing their statistical significance. A clear example is the use of comparative questions, which probed the participants for their opinions on the extent to which the MLA-Helper's visualization tool achieved a specific result compared to the other tools used for the test. These questions aimed to assess participants' opinions on the metrics regarding MLA-Helper's visualization feature compared to the other state-of-the-art tools. Unfortunately, testing statistical significance between these types of questions does not provide any benefit towards this assessment. A statistical significance in the difference between the results of two comparative questions would only prove that opinions on the two comparisons differ. Information on which element of the comparison satisfies the characteristic mentioned by the questions can only be inferred from the two resulting medians, but without any statistical proof of significance. A better approach, which was only adopted for some evaluated metrics, would have been to pose the same question regarding a specific characteristic once per each compared tool. This way, by testing the statistical difference of question pairs through paired tests, information could be gathered on how well each tool achieved the results.

Likert-type and Likert Scale Biases

Most of the answers to the Likert-type and Likert scale-type questions featured in the questionnaires failed to reject the null hypothesis of the Shapiro-Wilk test for normality (Samuel Shapiro & Martin Wilk, 1965), indicating a non-normal distribution. The reason behind the non-normality of the

samples can possibly be found in the small sample size and in Likert answers being ordinal, hence not normally distributed by definition (Alexander Robitzsch, 2020). This finding and the ordinal and interval nature of the measurements yielded by Likert questions led to the choice of the Wilcoxon Signed-Rank test for assessing the statistical significance of the findings (Frank Wilcoxon, 1945). The reasoning behind this choice can be found in the assumptions of the Wilcoxon Signed-Rank test. The first assumption is that the two samples are related to a dependent variable. The second assumption is that the two samples are randomly and independently drawn. The third assumption is that the measurements present a non-normal distribution. The fourth and final assumption is that the measurements are of the interval type, or at least the ordinal type. Some of the questions present in the questionnaires, as it will be later pointed out, will fulfill the requirements stated by these assumptions, justifying the choice of the test. Data was considered of the ordinal type when coming from Likert-type question results and interval type when coming from Likert scale question results, as suggested by Boone Jr. & Boone. Considering Likert data as ordinal or interval type has drawbacks, especially when applying statistical significance tests. Both ordinal and interval data are assumed to have a natural order, while that is not truly the case with Likert data. Notably, the central value of the scale should not be included in the ordering of the measures, as its meaning is that of neutrality rather than of a measure of opinion, as defined by the Likert scale's specification. Nonetheless, the Wilcoxon Signed-Rank test was proved to be robust for testing statistical significance for Likert data with a non-normal distribution (Gary E. Meek et al., 2007).

Model Visualizer Evaluation

The first evaluation was conducted to assess the explainability, clarity, intuitiveness, and ease of use metrics for the visualization tool of the MLA-Helper suite. The following are the results regarding the main questions that assessed these metrics.

Normality

Firstly, the normality of the results was tested via the Shapiro-Wilk test. As shown in Table 1, the null hypothesis was rejected for each question, meaning that none of the results for the questions proposed in the questionnaire presented a normal distribution.

Table 1

Results of the Shapiro-Wilk test of normality for the visualization tool's questionnaire quantitative data.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Question 1	,273	21	<,001	,774	21	<,001
Question 2	,372	21	<,001	,633	21	<,001
Question 3	,397	21	<,001	,620	21	<,001
Question 4	,446	21	<,001	,570	21	<,001
Question 5	,309	21	<,001	,832	21	,002
Question 6	,276	21	<,001	,765	21	<,001
Question 7	,309	21	<,001	,832	21	,002
Question 8	,248	21	,002	,874	21	,012
Question 9	,264	21	<,001	,820	21	,001
Question 10	,382	21	<,001	,689	21	<,001
Question 11	,290	21	<,001	,800	21	<,001
Question 12	,397	21	<,001	,620	21	<,001
Question 13	,315	21	<,001	,840	21	,003
Question 14	,232	21	,004	,872	21	,010
Question 15	,427	21	<,001	,646	21	<,001
Question 16	,454	21	<,001	,531	21	<,001

a. Lilliefors Significance Correction

Note. The last column shows the p-values for the questions, all lower than the significant value of 0.05.

Explainability Assessment

As stated at the start of the 'Evaluation' chapter, Zhou et al. describe explainability as a composite metric that includes the concepts of interpretability (degree of understanding) and clarity (unambiguity of the explanation).

Interpretability. For the analysis of interpretability, the following quantitative questions were chosen as the prominent representatives of the assessment:

1. The visualization I was presented with using MLA-Helper was more comprehensible than the visualization presented in Zetane.
2. The visualization I was presented with using MLA-Helper was more comprehensible than the visualization presented in Netron.
3. I felt MLA-Helper's visualization of the model's structure matched the mental image I had in my mind more than when using Zetane.
4. I felt MLA-Helper's visualization of the model's structure matched the mental image I had in my mind more than when using Netron.

The questions focused on analyzing the same concept, collocating them into the Likert scale-type question group. Following the procedure described in the premise, this characteristic translates to their mean and standard deviation analysis.

Table 2

Descriptive values for the interpretability-focused questions.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 1	21	1	3	2,33	,658
Question 2	21	3	4	3,57	,507
Question 5	21	2	5	3,48	,750
Question 6	21	2	5	4,19	,928
Valid N (listwise)	21				

As shown in Table 2, the standard deviations of the four questions are low, indicating that the majority of the participants shared opinions regarding interpretability. The highest mean is found in the responses to the fourth question, regarding how the model's structure visualization matched the mental image of the participants when using MLA-Helper compared to Netron. Conversely, the lowest mean is

found in the responses to the first question regarding how the model's visualization was more comprehensible in MLA-Helper compared to Zetane. The remaining questions present a mean close to the neutrality value, which could be associated with confusion regarding the questions themselves, not allowing for further interpretations. Overall, the means show some positive and negative opinions regarding the interpretability aspects of MLA-Helper compared to the other two tools. Unfortunately, since questions comparing Zetane and Netron on the same aspects were missing from the questionnaire, no assessment of the level of interpretability can be made through these results. However, some findings can still be presented through the analysis of the answers to the qualitative questions during the same evaluation. When asked to list the elements that helped the most in understanding the model's structure, most participants mentioned the color-coding and action mask visualization present in MLA-Helper's visualization tool. These were the most common elements when answering the question regarding differences found with the other state-of-the-art tools. These findings direct to several valuable interpretability elements in the presented suite's visualization tool. Even so, the overall interpretability of MLA-Helper's visualization tool still needs to be improved when compared to tools such as Zetane.

Clarity. For the analysis of clarity the following questions were chosen instead:

9. Certain aspects of the visualization were unclear or confusing when using MLAHelper.
10. Certain aspects of the visualization were unclear or confusing when using Zetane.
11. Certain aspects of the visualization were unclear or confusing when using Netron.

As with the interpretability evaluation, the questions considered for clarity focused on analyzing the same concept, collocating them into the Likert scale-type question group. This translates to the analysis of their mean and standard deviation being the key aspects of the evaluation.

Table 3

Descriptive values for clarity-focused questions.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 9	21	1	5	3,43	,870
Question 10	21	1	3	1,43	,598
Question 11	21	3	5	3,95	,669
Valid N (listwise)	21				

As shown in Table 3, the standard deviations of the four questions are low, indicating that the majority of the participants shared opinions regarding interpretability. Given the lowest mean of 1.43 found in the data of the second question and a much higher mean in the first question, it is apparent that MLA-Helper's visualization tool lacked clarity when compared to Zetane. The opposite consideration could be stated about the clarity of the presented suite when compared to Netron, given the higher mean in the third question. However, to prove these differences' statistical significance, a Wilcoxon Signed-Rank test was carried out for the two question pairs (Question 9 & Question 10, and Question 9 & Question 11). The Wilcoxon Signed-Rank test was applicable because the pairs of questions focus on the same dependent variable (clarity) while having an independent variable change (tool used). Table 4 and Table 5 show the results of the two tests.

Table 4

Results of the Wilcoxon Signed-Rank test for the pair Question 9 & Question 10.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The median of differences between Question 9 and Question 10 equals 0.	Related-Samples Wilcoxon Signed Rank Test	<,001	Reject the null hypothesis.

a. The significance level is ,050.

b. Asymptotic significance is displayed.

Note. Given the Z-value lower than 0.05 the null hypothesis was rejected.

Table 5

Results of the Wilcoxon Signed-Rank test for the pair Question 9 & Question 11.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The median of differences between Question 9 and Question 11 equals 0.	Related-Samples Wilcoxon Signed Rank Test	,064	Retain the null hypothesis.

a. The significance level is ,050.

b. Asymptotic significance is displayed.

Note. Given the Z-value higher than 0.05 the null hypothesis was retained.

The results from the test show a statistical significance only in the pair composed of Question 9 and Question 10. This finding validates the difference in clarity between MLA-Helper's visualization tool and Zetane but fails to validate the difference between the former and Netron. However, trends from the qualitative answers regarding the question "Were there any specific visualizations that you found particularly unclear or confusing?" still help give insights into the MLA-Helper's visualization tool clarity assessment. Almost all of the participants answered this question, mentioning how MLA-Helper's visualization tool portrays the layers and their information, signifying an overall acknowledgment from the participants regarding the overall lack of clarity of the visualization.

Intuitiveness Assessment

To evaluate the intuitiveness of MLA-Helper's visualization tool, the following questions from the quantitative questionnaire were chosen:

3. I found MLA-Helper's setup process more intuitive than Zetane's.
4. I found MLA-Helper's setup process more intuitive than Netron's.

For the same reasons explained in the previous assessments, these questions will be considered Likert scale-type questions that focus on intuitiveness. The following is the analysis of the mean and standard deviation of the data gathered from the participants' answers to the questions.

Table 6

Descriptive values for the intuitiveness-focused questions.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 3	21	1	2	1,38	,498
Question 4	21	1	2	1,71	,463
Valid N (listwise)	21				

As shown in Table 6, the standard deviations are both low, indicating a shared opinion of the participants on the matter. The means are also closer to the lowest value of the Likert scale, indicating an almost complete disagreement on the presented visualization tool's intuitiveness being higher than the one offered by the state-of-the-art tools. This finding proves that MLA-Helper's visualization tool failed to achieve a satisfactory result in the intuitiveness metric of the problem statement.

Ease of use Assessment

When evaluating the ease of use provided by MLA-Helper's visualization tool, the following representative quantitative questions were chosen from the questionnaire:

12. I felt the design of MLA-Helper's interface was easy to navigate.
13. I felt the design of Zetane's interface was easy to navigate.
14. I felt the design of Netron's interface was easy to navigate.

These questions will be considered Likert scale-type questions that focus on the concept of ease of use. The following analysis will observe the means and standard deviations of their answers.

Table 7

Descriptive values for the questions focused on ease of use.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 12	21	4	5	4,38	,498
Question 13	21	2	5	3,19	,750
Question 14	21	2	5	3,24	,831
Valid N (listwise)	21				

As shown in Table 7, the standard deviations indicate a highly shared opinion among the participants, and the fact that the mean of Question 12 is higher than the other two indicates that participants considered the ease of use of MLA-Helper's visualization tool to be higher than both Zetane's and Netron's. To test the validity of this assumption, a Wilcoxon Signed-Rank test was run on Question 12 and Question 13, and Question 12 and Question 14. The test was applicable given that both pairs of questions focused on the same dependent variable (ease of use) while having an independent variable change (tool used). Table 8 and Table 9 show the results of the two tests.

Table 8

Results of the Wilcoxon Signed-Rank test for the pair Question 12 & Question 13.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The median of differences between Question 12 and Question 13 equals 0.	Related-Samples Wilcoxon Signed Rank Test	<,001	Reject the null hypothesis.

a. The significance level is ,050.

b. Asymptotic significance is displayed.

Note. Given the Z-value lower than 0.05 the null hypothesis was rejected.

Table 9

Results of the Wilcoxon Signed-Rank test for the pair Question 12 & Question 14.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The median of differences between Question 12 and Question 14 equals 0.	Related-Samples Wilcoxon Signed Rank Test	<,001	Reject the null hypothesis.

a. The significance level is ,050.

b. Asymptotic significance is displayed.

Note. Given the Z-value lower than 0.05 the null hypothesis was rejected.

The results from the test show a statistical significance in the difference between the means of both pairs. This finding validates the assumption regarding participants' opinion on MLA-Helper's ease of use being higher than both state-of-the-art tools, achieving a valuable result towards the goal set by the problem statement.

Workflow Evaluation

The evaluation of the MLA-Helper's workflow aimed to assess the metrics of debugging capabilities and usability defined by the problem statement. The values of the answers the participants gave related to the metrics when using ML-Agents alone compared to using ML-Agents and MLA-Helper together. Unfortunately, it was later addressed that to gain a statistically valid insight into the differences between the interactions (ML-Agents alone and ML-Agents & MLA-Helper), the questionnaire should have been given to the participants at the end of each interaction, with the question being split into two separate ones addressing the two interaction cases separately. By applying this change, the answers could have been considered and tested with a paired test that would have proven the statistical significance of the difference between the mean of the answers given after the interaction with ML-Agents alone and the mean of the answers given after the interaction with ML-Agents and MLA-Helper. Nonetheless, some statistical elements will still be observed even though their validity is not proven.

Debugging Capabilities Assessment

Normality. Firstly , the normality of the results was tested via the Shapiro-Wilk test. As shown in Table 10, the null hypothesis was rejected for each question, meaning that none of the results for the questions proposed in the questionnaire presented a normal distribution.

Table 10

Results of the Shapiro-Wilk test of normality of the workflow questionnaire quantitative data.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Question 1	,251	15	,012	,799	15	,004
Question 2	,212	15	,068	,817	15	,006
Question 3	,385	15	<,001	,630	15	<,001
Question 4	,238	15	,022	,817	15	,006

a. Lilliefors Significance Correction

Note. The last column shows the p-values for the questions, all lower than the significant value of 0.05.

To assess the debugging capabilities, the answers to the following questions were analyzed:

1. Using MLA-Helper significantly improved my ability to debug ML-Agents projects compared to using ML-Agents alone.
2. I feel more confident in my ability to debug ML-Agents projects with the assistance of MLA-Helper.
3. MLA-Helper's debugging tools and features are more effective than those available in ML-Agents.
4. Using MLA-Helper was not helpful when debugging ML-Agents projects.

Given that all the above questions relate to the debugging capabilities when using ML-Agents with or without MLA-Helper, they will be considered Likert scale-type questions. The following is the analysis of their meaningful statistical values.

Table 11

Descriptive values for the questions related to the debugging capabilities.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 1	15	3	5	4,13	,834
Question 2	15	3	5	3,93	,799
Question 3	15	4	5	4,60	,507
Question 4	15	1	3	1,87	,743
Valid N (listwise)	15				

As portrayed in Table 11, the low standard deviation for all four questions indicates a shared opinion on the different topics among the participants. The high means of the first three questions indicate a shared opinion on the overall positive effect of MLA-Helper usage on the debugging capabilities when using it in combination with ML-Agents. Given the negative nature of the question, the low mean on the last question consolidates the opinion. However, as stated at the start of the 'Workflow Evaluation' subchapter, the statistical significance of the difference in the opinions relating to the debugging capabilities when using ML-Agents alone compared to using ML-Agents in combination with MLA-Helper can not be proven. Additionally, statistical significance could have also been tested taking into account the membership of the participants to the control group and experimental group detailed in the ‘Workflow Evaluation’ subchapter, but the invalid nature of the measurements prevented its application. Nonetheless, the opinion is still reinforced by the qualitative answers to the questionnaire. Particularly, the majority of participants, when asked if they noticed any useful debugging in MLA-Helper, answered by listing multiple elements, such as the possibility of setting constant observations and the action mask editor. Conversely, when asked to state issues in debugging with MLA-Helper (if any), the majority of participants did not fill in the answer.

Usability Assessment

Usability was assessed using the System Usability Scale, measuring the answers of the participants with a Likert scale that ranges from 1 to 5 (1 meaning Strongly Disagree and 5 meaning Strongly Agree). The following are the questions present in the questionnaire:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

Before moving to the calculation of the SUS overall score for addressing the MLA-Helper suite's usability, it is noteworthy that the SUS questionnaire can be considered as a list of Likert scale-type questions. This allows for statistical observations to be made on their means and standard deviations.

Table 12

Descriptive values for the SUS questionnaire's questions.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 1	15	3	5	3,93	,458
Question 2	15	1	5	2,80	1,656
Question 3	15	3	5	3,93	,594
Question 4	15	1	2	1,07	,258
Question 5	15	4	5	4,33	,488
Question 6	15	1	2	1,33	,488
Question 7	15	4	5	4,20	,414
Question 8	15	1	2	1,40	,507
Question 9	15	3	4	3,33	,488
Question 10	15	2	4	3,00	,756
Valid N (listwise)	15				

Table 12 shows the means for each SUS question and their standard deviations. Question 2, in particular, presents an unusually high standard deviation compared to the other questions. This translates to the participants being highly divided on the question regarding the system complexity, with some of the minimum and maximum values of the scale included in the answers. The reason behind this standard deviation could possibly be found in the difficulty of assessing the overall complexity of a suite such as MLA-Helper, being that it is composed of different tools with different functions.

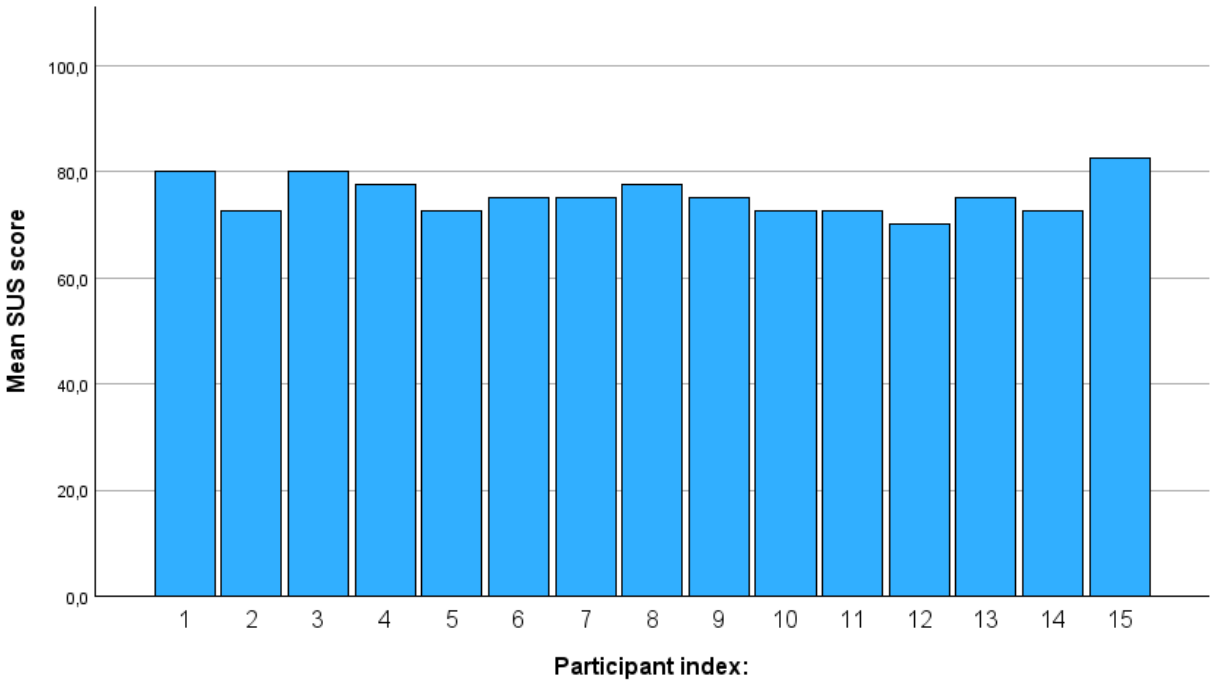
SUS score(Heading Level 4). As described by Brooke,

> To calculate the SUS score, first sum the score contributions from each item. Each item's score contribution will range from 0 to 4. For items 1, 3, 5, 7,and 9 the score contribution is the scale position minus 1. For items 2 ,4, 6, 8 and 10, the contribution is 5 minus the scale position. Multiply the sum of the scores by 2.5 to obtain the overall value of SU. SUS scores have a range of 0 to 100.
>
> (Brooke, 1996)

The following are the SUS scores from all the participants.

Figure 11

Mean SUS score for each participant.



As shown by Figure 11, each participant's mean of the SUS scores is close to 75, with a minimal standard deviation. Following the usage guidelines for SUS score interpretation, the usability of the MLA-Helper suite is considered to be medium-high, with its mean of the SUS score almost being one standard deviation away from reaching the threshold for the highest level of usability at 80.3 SUS score. Given these results, the usability of MLA-Helper can be positively assessed, fulfilling the objective of the problem statement. That said, the high score might have been influenced by the close connection with the participants given by the employment of the snowball method. An influencing factor might have also been the confirmation bias of the participants, trying to answer positively to align with the developer's expectations. Since the results were validated by the use of the SUS questionnaire and given the measurements of the answers of ordinal non-normal nature, a Mann-Whitney U test was run to prove the statistical significance in the difference between the control group's answers and the experimental

group's answers (Whitney & Mann, 1947). The following table presents the result of the test, considering the control group as Group 1 and the experimental group as Group 2.

Table 13

Results of the Mann-Whitney U test for the SUS questions taking into account the two participant groups.

Hypothesis Test Summary

	Null Hypothesis	Test	Sig.[a,b]	Decision
1	The distribution of SUS score is the same across categories of Group.	Independent-Samples Mann-Whitney U Test	,397[c]	Retain the null hypothesis.

a. The significance level is ,050.

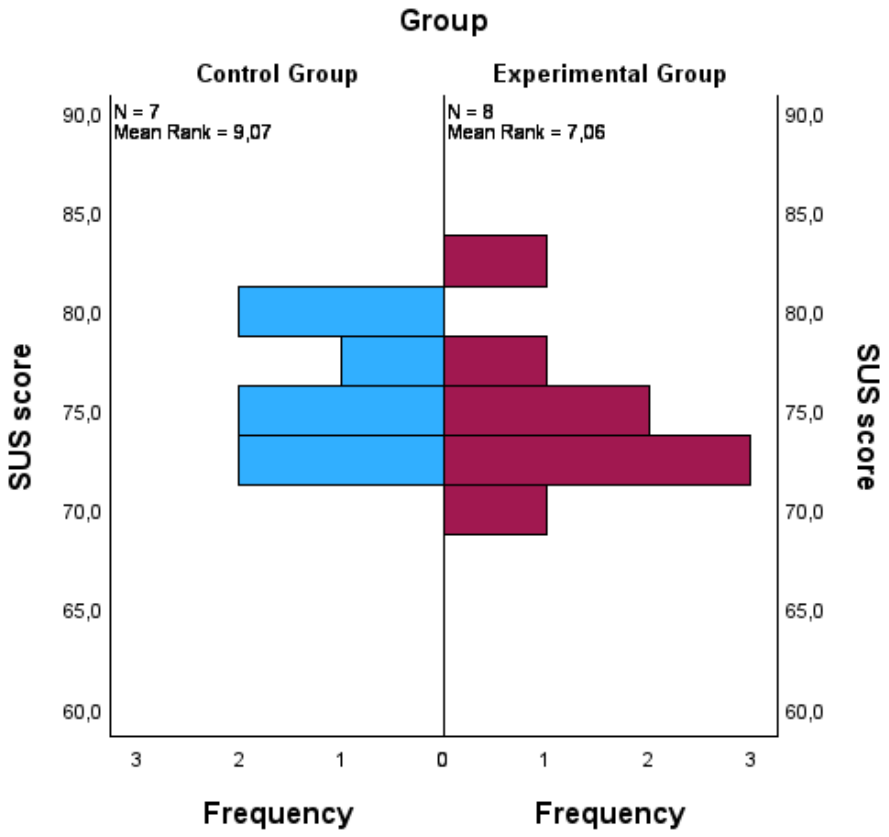
b. Asymptotic significance is displayed.

c. Exact significance is displayed for this test.

Note. Given the p-value lower higher 0.05 the null hypothesis was retained.

Figure 12

Frequency distributions of the mean SUS scores for the two participant groups.



As shown in Table 13, the null hypothesis is retained, signifying no statistical difference between the distributions of SUS scores across the two groups. Additionally, Figure 12 shows the distribution of their frequencies, to gain a visual insight into their similarity.

Performances Evaluation

In order to assess the overall performance of the MLA-Helper suite, two evaluations were carried out: an evaluation of its visualization tool's performance and an evaluation of its workflow performance. The first aimed at assessing the metrics defined in the problem statement: rendering time, rendering performance, and memory consumption. These measurements were gathered in MLA-Helper, Zetane, and Netron to compare the presented suite's performance to similar state-of-the-art visualization tools.

The second evaluation focused on assessing the metrics of debugging capabilities and usability for the presented suite's workflow by analyzing the time the participants took to set up the rock-paper-scissors game and to debug the card game.

Model Visualizer's Performances

The evaluation of the visualization tool's performances produced measurements regarding the average time before the model is rendered on screen, the average frames-per-second reached during the visualization, and the average memory consumption of the tool when the visualization is running. Unfortunately, the visualization performance analysis was rendered invalid due to inconsistent readings caused by the influence of other processes during the tests and fluctuations in the overall measurements. An utterly subjective opinion regarding performance drawbacks could still be inferred from the qualitative answers of the other evaluations. When asked to list issues noticed during their interaction with the MLA-Helper suite, the participants never mentioned performance-related issues. This may be influenced by the size of the models the users were presented with, which can be considered small on average.

Workflow's Performances

In this evaluation, the participants were divided into two groups based on the sequence in which they interacted with ML-Agents alone and ML-Agents and MLA-Helper together. This division was intended to assess the statistical significance of the performance difference between the two groups. To analyze the results, the sequence that presents the participants with ML-Agents alone first will be named Sequence One, and the opposite sequence will be named Sequence Two. The following are the descriptive values for the measurements of both groups.

Table 14

Descriptive values for MLA-Helper's workflow performance divided by interaction sequence.

Descriptive Statistics

Sequence		Time to setup rock-paper-scissor without MLA-Helper (seconds)	Time to setup rock-paper-scissor with MLA-Helper (seconds)	Time to debug card game without MLA-Helper (seconds)	Time to debug card game with MLA-Helper (seconds)
Without MLA-Helper first	Mean	402,29	310,43	532,43	170,29
	N	7	7	7	7
	Std. Deviation	107,647	65,053	77,235	34,548
	Minimum	238	223	398	116
	Maximum	512	379	600	206
With MLA-Helper first	Mean	336,88	371,25	213,63	368,63
	N	8	8	8	8
	Std. Deviation	43,943	47,865	30,645	54,290
	Minimum	274	315	153	289
	Maximum	402	472	246	443
Total	Mean	367,40	342,87	362,40	276,07
	N	15	15	15	15
	Std. Deviation	84,099	62,815	173,577	111,694
	Minimum	238	223	153	116
	Maximum	512	472	600	443

Table 14 shows a generally low standard deviation across the measurements, given the nature of the metrics analyzed. The only high standard deviation is found in the data regarding the time needed to set up the rock-paper-scissors game without MLA-Helper for the group of participants that interacted with ML-Agents alone first. This finding is possibly influenced by the participants' varying knowledge of the programming notions needed to set up an agent's training in ML-Agents. Additionally, the means of the measurements relative to the interaction without MLA-Helper are overall lower than the ones relative to the interaction with MLA-Helper, signifying an increase in debugging capabilities and usability given by the usage of MLA-Helper. To prove the statistical significance of this finding, the normality of the data must be tested first.

Table 15

Results of the Shapiro-Wilk test of normality for the MLA-Helper's workflow performance data.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	df	Sig.	Statistic	df	Sig.
Time to setup rock-paper-scissor without MLA-Helper (seconds)	,159	15	,200*	,925	15	,230
Time to setup rock-paper-scissor with MLA-Helper (seconds)	,155	15	,200*	,953	15	,566
Time to debug card game without MLA-Helper (seconds)	,282	15	,002	,832	15	,010
Time to debug card game with MLA-Helper (seconds)	,201	15	,103	,919	15	,184

*. This is a lower bound of the true significance.

a. Lilliefors Significance Correction

Note. The last column shows the p-values of the questions, all lower than the significant value of 0.05.

In Table 15, the Shapiro-Wilk test results are shown for each measurement, indicating a normal distribution for the measurements of the setup time of the rock-paper-scissors game. On the other hand, the data gathered during the debugging tests presents a non-normal distribution when the card game was debugged without MLA-Helper and a normal distribution when MLA-Helper was used. This prevents both parametric and non-parametric paired tests from being run on the debugging performance data, given that either of the two variables would not meet the distribution assumptions of the tests. The statistical significance of the differences in the means of the data regarding the two setup interactions was tested using a paired t-test (Student, 1908). This test was chosen because the subjects were independent, the fact that measurements were gathered from each participant, and the data distribution being normal. The following table shows the results of the test.

Table 16

Results of the Student's paired t-test on the data related to the set up task.

Paired Samples Test

	Paired Differences					Significance	
	Mean	Std. Deviation	Std. Error Mean	t	df	One-Sided p	Two-Sided p
Time to setup rock-paper-scissor without MLA-Helper (seconds) - Time to setup rock-paper-scissor with MLA-Helper (seconds)	24,533	76,867	19,847	1,236	14	,118	,237

Note. The last column shows the two-sided p-value resulting from the test, being higher than the significant value 0.05, thus retaining the null hypothesis.

By analyzing the results of the tests shown in Table 16, it is clear that the difference in means between the time to set up the rock-paper-scissors game with and without MLA-Helper does not hold a statistical significance, being that the two-sided p-value is higher than the required 0.05. This observation invalidates the observations made on the means of the data gathered during the performance evaluation for the workflow. Unfortunately, due to this invalidation, the statistical significance of the differences in the means of the data regarding the group that followed Sequence One and the group that followed Sequence Two also loses importance and will not be analyzed.

Feature Ranking Evaluation

Finally, the last results that will be interpreted relate to the evaluation of the feature ranking tool. As mentioned multiple times throughout the paper, the suite's feature ranking was never completed, and an evaluation was run to gather data on its hypothetical usefulness and the confidence it would give its users by confirming their knowledge of essential elements in the environments they created. The questionnaire given to the participants presented the following quantitative questions:

1. I believe the feature ranking tool would help me better understand how my models view their environments.

2. I believe the feature ranking tool would be redundant because I already know the important elements in my games for achieving optimal results.
3. If I were to use machine learning, I would use this feature ranking tool to help me debug the behaviors of the agents.
4. The insights from the feature ranking tool increase would increase my confidence in making informed adjustments to the agent's input features.
5. The feature ranking tool's insights enhance my overall confidence in grasping the key factors influencing the agent's actions within its environment.

The first three questions can be considered Likert scale-type questions, measuring the participant's opinion on usefulness, given their focus on explainability, redundancy, and debugging capabilities. The last two questions can also be considered Likert scale-type questions, measuring the participant's opinion on the confidence given using the tool. Because of this finding, the questions' means and standard deviations will be used to interpret their results.

Table 17

Descriptive values for the questions regarding the feature ranking tool.

Descriptive Statistics

	N	Minimum	Maximum	Mean	Std. Deviation
Question 1	15	2	5	3,60	1,056
Question 2	15	1	5	2,33	1,234
Question 3	15	2	5	3,27	,961
Question 4	15	2	5	3,13	,915
Question 5	15	2	5	3,47	,915
Valid N (listwise)	15				

As shown in Table 17, the standard deviations of the overall data are considerably high. This possibly correlates to the difficulty of forming opinions on hypothetical scenarios rather than hands-on experiences. Given that the means of answers for the presented questions are all close to the neutrality value, no further assessment can be made on the metrics mentioned at the start of this subchapter. The

answers to the qualitative questions regarding unclear elements and limitations also proved to be generally confusing, showing that the participants did not understand the concept of the ranking tool itself.

Discussion

In the previous section, the evaluation results were detailed. This chapter will discuss the meaning of these findings in contrast to the problem statement's goals.

Project's Purpose

The main objective of this paper was to validate the presented MLA-Helper suite as a means for gathering explainability insights, improving usability and facilitating debugging utilities when using the ML-Agents package to train AI models.

Evaluation Overview

As described in the problem statement, the AI explainability attribute was evaluated through its interpretability and clarity subcomponents while including closely related concepts such as intuitiveness and ease of use. The evaluation further included assessments of the advantages of utilizing the suite for debugging purposes and its overall usability. Performance was also evaluated across diverse scenarios to assess the presented suite's behavior under different stress levels. Unfortunately, the overall results gathered from the evaluation process show a negative trend in most of the proposed assessments. The analysis showed a lack of validity in most cases. As such, AI explainability, initially intended as the suite's main focus point, was never fully achieved by any of the suite's elements. Similarly, the feature ranking evaluation resulted in data that signified mainly neutral opinions on its hypothetical usefulness and confidence given to users, ending in an irrelevant assessment. The evaluation of the visualization tool of the suite also provided insights into its lack of clarity, interpretability, and intuitiveness. The suite's ease of use was however found to be higher than similar state-of-the-art tools, suggesting that participants found it easy to approach and engage with. The results regarding MLA-Helper's workflow further showed

many positive opinions regarding its debugging capabilities, although at no statistical significance. Moreover, the workflow's usability was validated and proved to be well-received by the test group. These findings point towards MLA-Helper providing a good overall platform for engaging with ML-Agents and the model interface but falls short of providing a clear visualization and improving transparency of related AI. Overall, the presented suite failed to meet the objectives set by the problem statement while still featuring some elements that were positively recognized by the participants, particularly its usability, debugging capabilities, and ease of use.

Assessing Fluctuating Data and Implications

The fractured nature of MLA-Helper's evaluation results is reflected in the fact that the purpose of the suite has shifted throughout development. To reiterate, the suite's initial focus was the model visualization and the feature ranking, intended to increase AI transparency of the ML-Agents package. As the feature ranking was cancelled during development, the research scope was shifted elsewhere, which may be one of the underlying causes leading to the invalidation of the suite's explainability elements. As touched upon, data gathered during the evaluation process was, furthermore, in many cases, inconclusive or statistically insignificant. A possible reason can be found in the sample size used for the evaluation, which may have led to statistical tests failing to validate the assessed metrics. This may further be linked to a weak research motivation foundating the choice of research type, questionnaire structure, and posed questions. Although the chosen questionnaires relate concepts connected to the problem statement's metrics, these were not adequately motivated by related works and research. A better approach would have been to research studies on similar assessments and employ these as a foundation for both structures and, if adequate, questions.

Research Field

Comparing the suite with similar tools from the industry in the 'Related Works' chapter further demonstrates that the proposed suite needs to be revised in most aspects regarding explainability,

debugging, and training aid. ConvNetJS offers a layer-based visualization of the model's structure similar to MLA-Helper's but enhances its explainability through information related to data propagation in the network. Other explainability and debugging-centric tools, such as DeepVis, Harley's node-link visualization, the CAM visualizer by Zhou et al., and ActiVis's structure visualizer, offer more in-depth insights into the model's inner workings and allow for a broader range of debugging analyses. Furthermore, the state-of-the-art tools part of MLA-Helper's evaluation were superior when strictly considering explainability, reinforcing the statement on the presented suite's mediocrity in this area. As discussed, most industry visualization and debugging tools for DRL focus on CNN, translating into vastly different user needs from those of MLA-Helper. This further helps situate the perceived difficulty of the conducted evaluations within the field.

Future Research

While MLA-Helper's limitations are glaring, the space for future research is ample. Among elements discussed in the 'Shortcomings' sub-chapter of the 'Implementation' chapter, including a feature ranking tool in the MLA-Helper suite could be considered a vast improvement, as it would increase its potential in the AI explainability field. However, the design and purpose of such a tool would need to be revised with a focus on better clarity, as reflected in the feature ranking evaluation data. The visualization tool could also benefit from extensions of its visualization elements, where multiple layer-based visualizations in the industry allow inspection of each layer's neurons, links, and weights. Additionally, the suite's debugging-centric elements could be refined by adding active learning (CueFlik) and data flow visualization (ConvNetJS).

Conclusion

Throughout the thesis, the objective was to present and evaluate the MLA-Helper suite as a valid means for aiding users in training, visualizing, and debugging ML agents with Unity's ML-Agents package. The evaluation resulted in inconsistent and partially invalid data, providing insights into the tool's

highlights and flaws. The suite was found to be a poor candidate for AI explainability purposes, given the negative trends of the results of the explainability evaluation. Nonetheless, it proved to be a decent solution for increasing the usability of the ML-Agents package while also providing functional debugging-related elements. However, given the invalid performance assessment, its usability might still be nullified when using the suite in edge-case scenarios not covered by the evaluation. The overall results suggest that a future inclusion of new tools and features to better tackle the suite's explainability purpose would be beneficial. As stated in the 'Discussion' chapter, changes to the currently existing tools of the suite could also be employed to refine their workings and fit a broader group of use cases. Such improvements could include active learning, data flow visualization, in-depth layer information, and in-model methods. A better evaluation process could also be carried out in future suite releases to gather more meaningful data. Motivating the evaluation method choice through related studies on similar subjects could provide a valid assessment of the suite's characteristics. In light of the discussed results, the MLA-Helper suite can be viewed as a first step towards a better understanding of and interaction with creating, training, and analyzing AI in Unity.

References

Aamir, A., Tamosiunaite, M., & Wörgötter, F. (2022). Caffe2Unity: Immersive Visualization and Interpretation of Deep Neural Networks. *Electronics*, *11*(1), Article 1. https://doi.org/10.3390/electronics11010083

Alexander Robitzsch. (2020). *Why Ordinal Variables Can (Almost) Always Be Treated as Continuous Variables: Clarifying Assumptions of Robust Continuous and Ordinal Factor Analysis Estimation Methods*.

Alzubi, J. (2018). *Machine Learning from Theory to Algorithms: An Overview*. https://iopscience.iop.org/article/10.1088/1742-6596/1142/1/012012/pdf

Amershi, S., Fogarty, J., Ashish, K., & Desney, T. (2011). *Effective End-User Interaction with Machine Learning*.

Arya, V., Bellamy, R. K. E., Chen, P.-Y., Dhurandhar, A., Hind, M., Hoffman, S. C., Houde, S., Liao, Q. V., Luss, R., Mojsilović, A., Mourad, S., Pedemonte, P., Raghavendra, R., Richards, J., Sattigeri, P., Shanmugam, K., Singh, M., Varshney, K. R., Wei, D., & Zhang, Y. (2019). *One Explanation Does Not Fit All: A Toolkit and Taxonomy of AI Explainability Techniques* (arXiv:1909.03012). arXiv. http://arxiv.org/abs/1909.03012

Barracuda. (2018). https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html

Bellman, R. E. (1957). *Dynamic Programming*.

Ben-Kiki, O., & Evans, C. (n.d.). *YAML Ain't Markup Language (YAML™) Version 1.2*.

Boone Jr., H. N., & Boone, D. A. (2012). *Analyzing Likert Data*.

Brooke, J. (1996). *SUS -- a quick and dirty usability scale*.

Brooke, J. (2013). *SUS: A Retrospective*.

Chadi, M.-A., & Mousannif, H. (n.d.). *Understanding Reinforcement Learning Algorithms: The*

Progress from Basic Q-learning to Proximal Policy Optimization.

Chatzimparmpas, A., Martins, R. M., Jusufi, I., & Kerren, A. (2020). A survey of surveys on the use of visualization for interpreting machine learning models. *Information Visualization*, *19*(3), 207–233. https://doi.org/10.1177/1473871620904671

Choo, J., & Liu, S. (2018). Visual Analytics for Explainable Deep Learning. *IEEE Computer Graphics and Applications*, *38*(4), 84–92. https://doi.org/10.1109/MCG.2018.042731661

Christopoulou, E., & Xinogalos, S. (2017). Overview and Comparative Analysis of Game Engines for Desktop and Mobile Devices. *International Journal of Serious Games*, *4*(4). https://doi.org/10.17083/ijsg.v4i4.194

ConvNetJS. (2014). https://cs.stanford.edu/people/karpathy/convnetjs/

Dargan, S., Kumar, M., Ayyagari, M. R., & Kumar, G. (2020). A Survey of Deep Learning and Its Applications: A New Paradigm to Machine Learning. *Archives of Computational Methods in Engineering*, *27*(4), 1071–1092. https://doi.org/10.1007/s11831-019-09344-w

Doshi-Velez, F., & Kim, B. (2017). *Towards A Rigorous Science of Interpretable Machine Learning* (arXiv:1702.08608). arXiv. http://arxiv.org/abs/1702.08608

Fogarty, J., Tan, D., Kapoor, A., & Winder, S. (2008). CueFlik: Interactive concept learning in image search. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 29–38. https://doi.org/10.1145/1357054.1357061

Fradkov, A. L. (2020). Early History of Machine Learning. *IFAC-PapersOnLine*, *53*(2), 1385–1390. https://doi.org/10.1016/j.ifacol.2020.12.1888

Frank Wilcoxon. (1945). *Individual Comparisons By Ranking Methods*.

Furnkranz, J. (2001). *Machine Learning in Games: A Survey*.

Gary E. Meek, Ceyhun Ozgur, & Kenneth Dunning. (2007). *Comparison of the t vs. Wilcoxon*

Signed-Rank Test for Likert Scale Data and Small Samples.

GitHub. (n.d.). GitHub. Retrieved 30 August 2023, from https://github.com/

Goodman, L. A. (1961). *Snowball sampling*.

Harley, A. W. (2015). An Interactive Node-Link Visualization of Convolutional Neural Networks. In G. Bebis, R. Boyle, B. Parvin, D. Koracin, I. Pavlidis, R. Feris, T. McGraw, M. Elendt, R. Kopper, E. Ragan, Z. Ye, & G. Weber (Eds.), *Advances in Visual Computing* (Vol. 9474, pp. 867–877). Springer International Publishing. https://doi.org/10.1007/978-3-319-27857-5_77

Jepsen, P., Johnsen, S. P., Gillman, M. W., & Sørensen, H. T. (2004). Interpretation of observational studies. *Heart*, *90*(8), 956–960. https://doi.org/10.1136/hrt.2003.017269

Kahng, M., Andrews, P. Y., Kalro, A., & Chau, D. H. (2017). *ActiVis: Visual Exploration of Industry-Scale Deep Neural Network Models* (arXiv:1704.01942). arXiv. http://arxiv.org/abs/1704.01942

Kaynak, O. (2021). The golden age of Artificial Intelligence. *Discover Artificial Intelligence*, *1*(1), 1. https://doi.org/10.1007/s44163-021-00009-x

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2018). Feature Selection: A Data Perspective. *ACM Computing Surveys*, *50*(6), 1–45. https://doi.org/10.1145/3136625

Li, Y. (2018). *Deep Reinforcement Learning: An Overview* (arXiv:1701.07274). arXiv. http://arxiv.org/abs/1701.07274

Liu, S., Wang, X., Liu, M., & Zhu, J. (2017). Towards better analysis of machine learning models: A visual analytics perspective. *Visual Informatics*, *1*(1), 48–56. https://doi.org/10.1016/j.visinf.2017.01.006

McCloskey, B. (2022, November 29). *Deep Learning Model Visualization Tools: Which is Best?* Medium. https://towardsdatascience.com/deep-learning-model-visualization-tools-which-is-best-83ecbe14fa7

Metropolis, N., & Ulam, S. (1949). *The Monte Carlo Method*.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., Krishnamurthy, J., Lao, N., Mazaitis, K., Mohamed, T., Nakashole, N., Platanios, E., Ritter, A., Samadi, M., Settles, B., … Welling, J. (2018). Never-ending learning. *Communications of the ACM*, *61*(5), 103–115. https://doi.org/10.1145/3191513

ML-Agents Docs. (n.d.). GitHub. Retrieved 19 July 2023, from https://github.com/bascoul/Ml-Agents/blob/master/docs/Training-ML-Agents.md

Mousavi, S. S., Schukat, M., & Howley, E. (2018). *Deep Reinforcement Learning: An Overview* (Vol. 16, pp. 426–440). https://doi.org/10.1007/978-3-319-56991-8_32

Nader, A., & Azar, D. (2021). Evolution of Activation Functions: An Empirical Investigation. *ACM Transactions on Evolutionary Learning and Optimization*, *1*(2), 1–36. https://doi.org/10.1145/3464384

Nguyen, A., & Martínez, M. R. (2020). *On quantitative aspects of model interpretability* (arXiv:2007.07584). arXiv. http://arxiv.org/abs/2007.07584

Oladipupo, T. (2010). Types of Machine Learning Algorithms. In Y. Zhang (Ed.), *New Advances in Machine Learning*. InTech. https://doi.org/10.5772/9385

ONNX About. (n.d.). Retrieved 1 August 2023, from https://onnx.ai/about.html

ONNX Documentation. (n.d.). Retrieved 19 July 2023, from

https://onnx.ai/onnx/repo-docs/IR.html#nodes

ONNX Home. (n.d.). Retrieved 19 July 2023, from https://onnx.ai/

Patel, A. (2023). *Ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network* [Computer software]. https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network (Original work published 2019)

Pezzotti, N. (2017). *Dimensionality-Reduction Algorithms for Progressive Visual Analytics* [Delft University of Technology]. https://doi.org/10.4233/UUID:DF6C0760-89BA-4DB0-9621-19C512EB1955

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). *Searching for Activation Functions* (arXiv:1710.05941). arXiv. http://arxiv.org/abs/1710.05941

Roeder, L. (2023). *Lutzroeder/netron* [JavaScript]. https://github.com/lutzroeder/netron (Original work published 2010)

Rowley, J. (n.d.). Designing and using research questionnaires. *11 March 2014*.

Ryan Hipple (Director). (2017, November 20). *Unite Austin 2017—Game Architecture with Scriptable Objects*. https://www.youtube.com/watch?v=raQ3iHhE_Kk

Sadangi, S. (2022, July 29). *The Best Tools for Machine Learning Model Visualization*. Neptune.Ai. https://neptune.ai/blog/the-best-tools-for-machine-learning-model-visualization

Samuel Shapiro & Martin Wilk. (1965). *An Analysis of Variance Test for Normality (Complete Samples)*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms* (arXiv:1707.06347). arXiv. http://arxiv.org/abs/1707.06347

Shapley, L. (1951). *Notes on the n-Person Game, II: The Value of an n-Person Game*.

Singh, M. (2023). *NetRunner* [C#]. https://github.com/maderix/NetRunner (Original work published 2021)

Statistics Café. (n.d.). *How to Use the Likert Scale in Statistical Analysis*.

Student. (1908). *The Probable Error of a Mean*.

TensorBoard. (2023). [TypeScript]. tensorflow. https://github.com/tensorflow/tensorboard (Original work published 2017)

Tensorflow. (2021). TensorFlow. https://www.tensorflow.org/tensorboard/graphs?hl=it

Torres, J. (2021, September 24). *The Bellman Equation*. Medium. https://towardsdatascience.com/the-bellman-equation-59258a0d3fa7

Unity Engine. (n.d.). Unity. Retrieved 29 August 2023, from https://unity.com

Unity for Visual Studio Code. (2023). https://marketplace.visualstudio.com/items?itemName=visualstudiotoolsforunity.vstuc

Unity ML-Agents Toolkit. (2023). [C#]. Unity Technologies. https://github.com/Unity-Technologies/ml-agents (Original work published 2017)

Unity Technologies. (n.d.-a). *ML-Agents Overview—Unity ML-Agents Toolkit*. Retrieved 1 August 2023, from https://unity-technologies.github.io/ml-agents/ML-Agents-Overview/#training-methods-environment-agnostic

Unity Technologies. (n.d.-b). *Unity - Manual: ScriptableObject*. Retrieved 29 August 2023, from https://docs.unity3d.com/Manual/class-ScriptableObject.html

Unity Technologies. (n.d.-c). *Unity - Manual: The Inspector window*. Retrieved 5 August 2023, from https://docs.unity3d.com/Manual/UsingTheInspector.html

Unity Technologies. (2021). *2021 Gaming Report—Unity insights from 2020 and predicted trends for 2021*.

Unreal Engine. (n.d.). Unreal Engine. Retrieved 29 August 2023, from https://www.unrealengine.com/en-US

Van Den Haak, M. J., De Jong, M. D. T., & Schellens, P. J. (2003). *Retrospective vs. Concurrent think-aloud protocols: Testing the usability of an online library catalogue*.

Visual Studio Code. (2015, April). https://code.visualstudio.com/

Whitney, D. R., & Mann, H. B. (1947). *On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other*.

Wouter van Heeswijk. (2023, August 16). *Proximal Policy Optimization (PPO) Explained*. Medium. https://towardsdatascience.com/proximal-policy-optimization-ppo-explained-abed1952457b

Xu, F., Uszkoreit, H., Du, Y., Fan, W., Zhao, D., & Zhu, J. (2019). *Explainable AI: A Brief Survey on History, Research Areas, Approaches and Challenges* (pp. 563–574). https://doi.org/10.1007/978-3-030-32236-6_51

Yosinski, J. (2023). *Deep Visualization Toolbox* [Python]. https://github.com/yosinski/deep-visualization-toolbox (Original work published 2015)

Yu, R., & Shi, L. (2018). A user-based taxonomy for deep learning visualization. *Visual Informatics*, *2*(3), 147–154. https://doi.org/10.1016/j.visinf.2018.09.001

Zetane Viewer. (2023). [Python]. Zetane Systems. https://github.com/zetane/viewer (Original work published 2021)

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning Deep Features

for Discriminative Localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921–2929. https://doi.org/10.1109/CVPR.2016.319

Zhou, J., Gandomi, A. H., Chen, F., & Holzinger, A. (2021). *Evaluating the Quality of Machine Learning Explanations: A Survey on Methods and Metrics*.